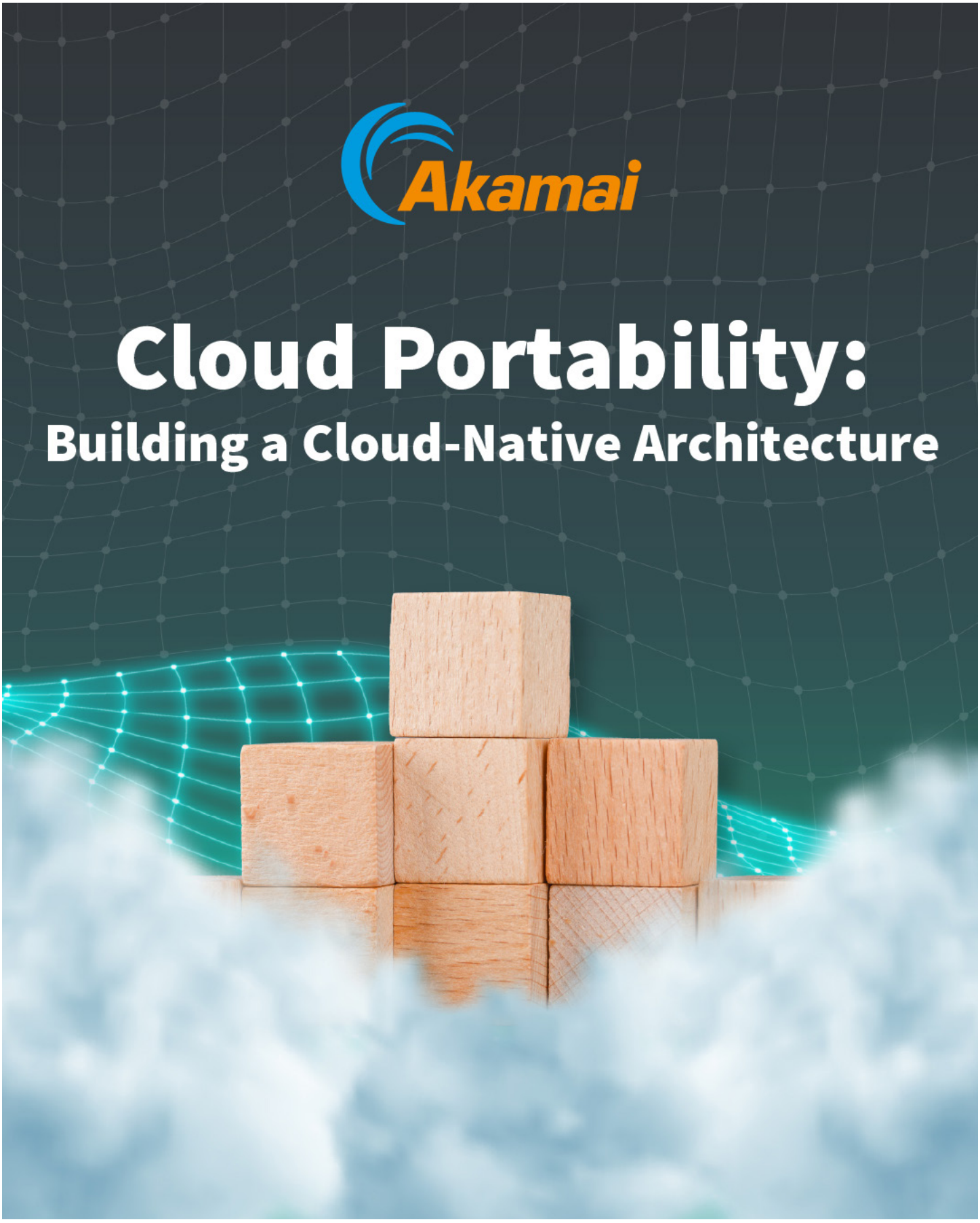


Akamai

EBOOK

CLOUD OVERVIEWS

Table of Contents

Introduction

A cloud vendor's job is to help their customers regain control of their infrastructure and gain the necessary flexibility and efficiency required of any developer today. Regardless of the provider, organizations should always take a best-of-breed approach when selecting a cloud vendor. At the same time, you should avoid locking yourself into a specific provider by utilizing proprietary tools.

Cloud providers should start by helping you make your workloads more portable across multiple clouds and your data more portable across different providers. And it's not just the technical side that should be a cloud provider's focus. How do they ensure your development talent and skills are portable across different clouds rather than being operationally locked into their tools and services?

Then there's cloud spend, which is increasingly getting out of control as different teams deploy resources that accumulate additional usage fees often not accounted for in the original budgeting process. Portability can play a significant role in reducing costs, helping to ensure you can move workloads across clouds to avoid getting locked into a single vendor.

> *Open source refers to licensing and distribution of software, while open standard refers to technical specifications or protocols that are developed and maintained in an open and collaborative way. Open source provides accessibility and sharing of source code, enabling collaborative software development, while open standards ensure interoperability and compatibility among different systems and technologies.*

Some cloud providers have an opinionated view of how you should build with them. By adopting a platform-centric approach, they want you to develop and deploy using *their* services and tooling within *their* ecosystem. But cloud providers shouldn't dictate how you build and deploy. Instead, workloads should be portable, using open- and standards-based tools, giving *you* the ability to deploy and move them where it makes sense for your workloads.

Developers Demand Portability

According to a TechStrong Research Summer 2023 survey of DevOps, cloud-native, and digital transformation professionals, two of the most popular use cases for workload portability were cost savings and resilience, following disaster recovery/business continuity. Latency, performance, and scalability also rated highly, which signals that respondents are looking at portability for optimization/efficiency as well.

It's clear that now is the time to reevaluate your cloud strategies and ensure your choices are best for your organization for the next ten years and beyond. While the traditional hyperscale cloud providers offer full-platform capabilities and enterprise-class benefits, such as scale and resiliency, they often force you into using proprietary, platform-specific tools. Nearly half of the TechStrong Research survey respondents said that they are looking to portability not only to move workloads between providers but to move off of one provider to another. Cloud-native technologies are critical to providing the freedom to move workloads from a single platform.

By working with a cloud provider that focuses on a cloud-native approach, you gain access to services and tools that are not only accessible but encourage you to move beyond a cloud ecosystem that generally looks the same way today as it did a decade ago. Most cloud implementations get stuck in a single centralized cloud or a few data centers, with workloads duplicated across multiple regions. It's time this changed.

Cloud has always been about the potential to get your workloads closer to your customers while avoiding services that lock you into a cloud-specific architecture. A cloud-native approach encourages using platforms, tools, and services built on open source and open standards, for maintaining interoperability in dynamic environments.

This ebook explores cloud-native development using microservices, containers, and other architecture decisions that allow for development with portably open standard tools. We'll explain what each means and provide cloud-native examples that you can use instead of their platform-centric counterparts. You also will find examples of how you can implement those tools on cloud platforms built for the future of cloud-native development and not the past.

Chapter 1

Benefits of a Cloud Portable Architecture

It doesn't matter what cloud provider you use if the workload is architected to be portable. When designing with portability and standardization in mind, start by identifying vendor lock-in points or where there's potential for a cloud vendor to prevent you from moving to another provider.

For instance, designing around things like Kubernetes is not always enough. Think about other systems the Kubernetes cluster might be interacting with or need to interact with. Assess those requirements and design your architecture using open source solutions and core cloud infrastructure primitives, which you can find across any cloud provider.

Focus on standard APIs to ensure compatibility between the application and other systems. Start with RESTful APIs for components that use synchronous (request/response) communication. Their popularity comes from using HTTP, the most common and widely-supported protocol. While other protocols have come and gone and will keep coming for things like video streaming, the ones that have stuck are HTTP-based.

Modular designs with microservices or containers break down the application into smaller, more manageable components, making it easier to add or remove functionality as needed and making the application more flexible and scalable. A cloud-native approach shines as it provides an efficient process for updating and replacing those components without affecting the entire workload.

Once you have a portable architecture, how do you keep it from being so overwhelming from a management point of view? Automation simplifies and streamlines the deployment and management of the application. Use continuous integration/continuous deployment (CI/CD) pipelines and infrastructure-as-code (IaC) tools.

A declarative approach to deployment allows you to codify every part of your workload: the application, software, and system configurations, everything it runs on in dev, staging, and production environments. As a result, you can quickly spin up on a new cloud, failover, or burst onto another cloud provider. With a fully-codified environment that is also versioned, you have everything documented regarding exactly how everything gets set up, and you have a transparent history of all of the changes and everyone who has made changes to it.

And, of course, you cannot forget about security. Following this path to portability allows you to develop a more standard approach to security, eliminating a very dangerous "set it and forget it" sort of mentality. This happens a lot: You deploy a workload on an isolated subnet in a virtual private cloud (VPC) and assume it's secure.

Within the same "Everything-as-Code" approach, you implement a portable, standardized architecture; aspects of your security posture can also be standardized and codified. In addition to DevSecOps practices that automate feedback loops, vulnerability testing, and so on, think about access control policies and hardened configuration templates; these can be unopinionated about the underlying platform and thus can secure your resources with consistency across environments. This approach to security can be immensely powerful, especially when you can pick up that application and drop it on any cloud provider. Being able to pick up your workload and move it around is a solid defensive strategy for disaster recovery.

Benefits of a Portable Architecture

A portable cloud architecture offers flexibility, cost optimization, resilience, scalability, and improved deployment practices. It empowers you to choose the best cloud solutions for your specific needs, avoid vendor lock-in, and adapt quickly to evolving requirements or market conditions. You can also:

Increase availability
A portable architecture ensures that applications can be easily deployed across multiple cloud platforms, reducing the risk of downtime due to platform-specific issues.

Improve agility
A cloud-native architecture enables developers to rapidly iterate and deploy new features and functionality, reducing time-to-market and improving competitiveness.

Improve security
A portable architecture enables applications and infrastructure components to be more consistently secured, with security features such as encryption and identity management integrated into the architecture instead of relying on platform features.

Increase resilience
A portable architecture ensures that applications can survive hardware and software failures without interruption or data loss.

Easier management
A portable architecture enables more efficient management of applications, with tools for monitoring, automation, and orchestration that work across multiple cloud platforms.

Increase innovation
A portable architecture enables organizations to use new and emerging technologies, such as AI and machine learning, to create innovative new applications and services.

Chapter 2

Microservices

Microservices should be scalable and focused on a single responsibility. Each self-contained modular unit handles a specific function within a larger system. A large application gets built from modular components or services like containers or serverless computing.

Think of your app as a business, and microservices as the departments, budgets, and requirements that comprise that business. Every year, these requirements change depending on the needs of the business. Your application also won't face the same level of demand over time. There could be some aspects that require more demand and others that you'll need to pay more attention to. Different levels of scaling also will need to occur within your application. Microservices allow you to scale and grow in different areas without affecting others. And they scale independently.

We all remember the single responsibility principle tenet from programming. Microservices aren't any different: They should do one thing and do that thing well. You also get the inherent benefit of better resilience and fault tolerance. Microservice architecture aims to prevent system-wide failures by oscillating faults to individual services. If there's a particular fault, you know where that is and can address it without impacting anything else. There's a discoverable aspect as well. By using a service networking/mesh solution like Consul from HashiCorp as an alternative to AWS App Mesh, you'll know when new services come online and have one centralized system that becomes a directory of services that defines what those services do and how to communicate with them.

Why You Should Consider Microservices

Faster time-to-market
Microservices enable parallel development and deployment of individual components, accelerating the overall development process and reducing the time it takes to deliver new features.

Improved scalability
Microservices can be scaled independently, allowing businesses to allocate resources more efficiently and handle varying workloads or traffic patterns more effectively.

Enhanced resilience
The decentralized nature of microservices reduces the risk of system-wide failures, ensuring continuous service availability and better overall system reliability.

Flexibility and adaptability
Microservices allow businesses to leverage diverse technologies and frameworks for different components, making it easier to adapt to changing requirements or incorporate new technologies.

Easier maintenance and updates
The modular design of microservices simplifies system maintenance and updates, as individual components can be upgraded or replaced without affecting the entire system.

Microservices Best Practices

Keeping microservices small, focused, and responsible for a single business capability is essential. This approach allows you to add additional functionality and avoid sprawl. However, there is no steadfast rule regarding the ideal size, as it will vary depending on the specific application and its requirements.

You also want to make sure you design for failure. While fault tolerance is built inherently into running multiple services and microservices by design, it adds additional resilience, such as retry mechanisms, circuit breakers, and bulkheads. Think of why ships have bulkheads. They have them for structural integrity, but they also have them if there's an issue, the bulkhead gets closed, and the ship doesn't sink. Many event-based architectures use what gets referred to as dead letter queues. If a message cannot get delivered, it goes into a particular queue where it can get inspected to determine the reason for failure.

Microservices should be designed based on domain-driven design principles, meaning modeling services based on business capabilities and using a common language to ensure services align with business needs. Domain-driven design focuses on creating software systems based on a deep business domain understanding. Its principles help guide the design process and ensure that the software aligns with the domain and provides value to the business. These principles collectively promote a deep understanding of the business domain and help ensure development remains closely aligned with the business needs and changing requirements.

Design with an API-first approach and implement API gateways, which provide central connecting points for facilitating communication between microservices and third-party subsystems. API gateways handle much of the routing, taking care of authorization, authentication, and rate limiting. A design pattern of APIs is essential for modularity and reusability of microservices.

Here are some additional microservices best practices:

Automate testing and deployment
Test and deploy microservices using automation tools such as continuous integration and continuous deployment (CI/CD) pipelines, which reduce the risk of errors ensuring services get deployed quickly and consistently.

Use containerization
Containerization provides a lightweight and portable way to package and deploy microservices. Using containerization can help simplify the deployment process and improve the application's scalability and portability.

Monitor and observe
Microservices should get monitored and logged to ensure they perform as expected and identify any issues or errors. Log aggregators and application performance monitoring (APM) tools can do this. Tracing provides insight into the flow of data through a distributed system. These three pillars help to provide end-to-end visibility over performance.

Secure services
Microservices should be secured using best practices such as authentication, authorization, and encryption, and don't forget container security! Policies should enforce which microservices can talk to others to reduce the overall attack surface. Security should be part of any design and checked throughout all phases of development, resulting in a far more secure application and protecting sensitive data.

Introduction to Continuous Integration and Continuous Delivery (CI/CD)

Introduction to Continuous Integration and Continuous Delivery (CI/CD)

What is Continuous Integration?

In software development, integration is the process of incorporating into the existing codebase changes like refactored code, new features, or bug fixes.

Traditionally, integration is performed in preparation of the deployment of a new release. This approach can lead to problems, as integrating multiple or large changes in one go may result in last-minute conflicts and bugs. Several strategies have emerged to mitigate these problems. Continuous Integration (CI) is one such solution: It creates a workflow where every commit made to the codebase is tested and built, and each developer on the project is expected to commit code at least once per day. Usually, both testing and building are automated, so that every commit to the central repository triggers an automatic build. If the build is successful, it triggers a run of the test suite. This way, conflicts or bugs are discovered quickly over the course of development, making the release process smoother and less painful.

What is Continuous Delivery?

Continuous Delivery (CD) takes the principle of CI one step further: The automated build and test process is extended to every component of the application, including configuration files, database schemas, and environments. Whenever changes are made to any of these components, the build and test procedure is performed (ideally through an automated pipeline) so that the resulting build can be easily deployed. The resulting build does not always have to be released to production, but it should at least be deployed to an environment (such as staging) that is as close to production as possible. Product owners can therefore be confident that any build that makes it through this pipeline is releasable to the public. This approach offers several advantages:

- New features and bug fixes can move from concept to deployment faster.
- Because the difference between each release is small, the risk of an individual deployment breaking is reduced.
- For the same reason, it is easier and safer to revert to a previous version in the event of an issue.
- As with continuous integration, continually building and testing environments and configuration parameters means that problems are discovered earlier in the development process.

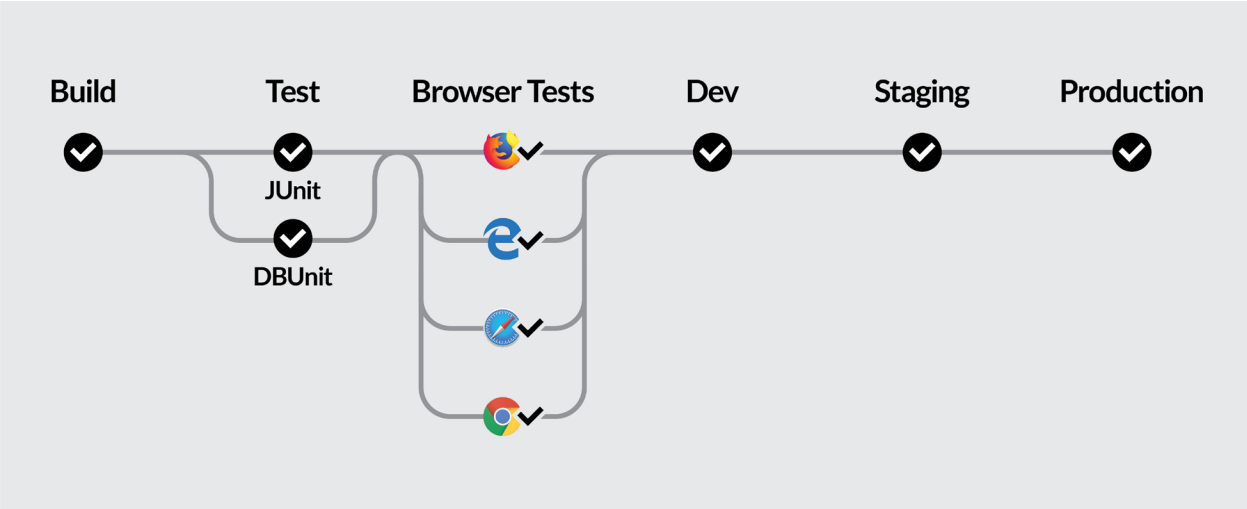
Principles of CI/CD

These principles are taken from the book Continuous Delivery by Jez Humble and David Farley.

1. **Create a repeatable, reliable process for deploying software:** If deploying and releasing software is easy and low-cost, it can be done more often and with greater confidence.
2. **Automate almost everything through the use of build and deploy scripts.**
3. **Keep everything in version control:** This makes it simple to revert to earlier versions, and provides a way to trigger the deployment pipeline as needed.
4. **Bring the pain forward:** The most painful and time-consuming aspects of development, which for most products includes testing, integration, and deployment, should be done more frequently and earlier in the process.
5. **Build quality in:** By fixing bugs and resolving problems earlier in the process, quality is built in continuously, rather than creating a broken product that requires time-consuming mitigation right before deployment.
6. **Done means released:** Since every increment in continuous delivery should be releasable (even if it is not actually released), no feature or task can be considered complete unless it can be released to users with the press of a button. In practice, since deployment is easy and cheap, many teams may consider a feature to be done only when it has been released to production.
7. **Everyone is responsible:** Every time a change is committed to version control, whoever commits the change is responsible for ensuring that the build completes and tests pass. In this way, members of all teams are responsible for keeping the project in a releasable state and preparing it for release.
8. **Continuous improvement:** Most of the work involved in a product will take place after the initial release, through maintenance, bug fixes, and added features.

Pipelines

Pipelines are an important concept in CI/CD. A pipeline is a sequence of steps that will be run whenever a change is made to a project (source code, database schemas, environments, or configuration). Pipelines can include steps executed in series or in parallel, and often multiple pipelines are used to handle different situations when working on a project.



This example illustrates a continuous delivery pipeline. When code is committed to the `staging` branch, it is built and tested on the CI server as before. This time, tests are run on different browsers for a more production-ready test environment. The environments are also dockerized, which helps guarantee that all developers will be working in identical environments and makes differences between production, staging, and test environments explicit. If these steps complete successfully, the build is then deployed to the staging branch.

Host a CI/CD Server on Linode

Usually, the integration pipeline runs on an external server rather than on developers' local computers. When developers commit code to the source repository, Git hooks are used to trigger the CI pipeline on a remote server, which then pulls the new code changes, builds the new version, and runs all of the tests against the new build. A Linode can be used as the remote server in this configuration, allowing you to set up whatever pipeline suits your projects's needs. See the links in the More Info section below for information on how to set up a CI server on your Linode.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

[How to Automate Builds with Jenkins on Ubuntu]

Open Standard Service Mesh Alternatives to AWS App Mesh

Copy and paste-able command line examples appear in the technical documentation here:

- Consul (HashiCorp)
- Istio (CNCF graduated project)
- Linkerd (CNCF graduated project)

Install HashiCorp Consul Service Mesh

Consul is a service mesh offered by HashiCorp, with robust service discovery and diagnostic features for managing your application's services. You can learn more about service meshes in our guide What Is a Service Mesh? Consul offers a balanced approach between flexibility and usability that makes it a compelling option for managing your service-oriented applications.

In this guide, you can see how to install and get started using the Consul service mesh with a Kubernetes cluster. You can get started with Kubernetes with our Linode Kubernetes Engine (LKE).

Before You Begin

- Follow the Linode Kubernetes Engine - Get Started guide to create a Kubernetes cluster using LKE.
- Make sure you install kubectl on your local machine and download your cluster's kubeconfig file.

Setting Up Consul

This section shows how to install and start using Consul with your Kubernetes cluster. By the end, you can have Consul deployed to your cluster and learn how to access its dashboard interface for monitoring services.

Install Helm

Helm is the standard method for installing Consul with Kubernetes. The details in the steps below assume a local machine running Linux on an AMD64 processor. However, they also work for macOS and other processors with slight modifications. For more details on the installation process, refer to Helm's official installation instructions.

1. Change into your current user's home directory, and download the `tar.gz` containing the Helm binary.

    ```
    cd ~/
    sudo wget https://get.helm.sh/helm-v3.6.1-linux-amd64.tar.gz
    ```

2. Extract the archive.

    ```
    sudo tar -zxvf helm-v3.6.1-linux-amd64.tar.gz
    ```

3. Move the Helm binary to a directory in your system path.

    ```
    sudo mv linux-amd64/helm /usr/local/bin/helm
    ```

Install Consul

1. Add the HashiCorp repository to Helm.

```
helm repo add hashicorp https://helm.releases.hashicorp.com
```

```
"hashicorp" has been added to your repositories
```

2. Verify that you have access to the Helm chart for Consul.

```
helm search repo hashicorp/consul
```

```
NAME            CHART VERSION APP VERSION DESCRIPTION
hashicorp/consul 0.32.0       1.10.0      Official HashiCorp Consul Chart
```

3. Update the repository.

```
helm repo update
```

```
Hang tight while we grab the latest from your chart repositories...
...Successfully got an update from the "hashicorp" chart repository
Update Complete. *Happy Helming!*
```

4. Create a configuration file for Consul. The parameters need to be adjusted for your needs. You can refer to our the example configuration file below (example-consul-config.yaml) for a basic working set of options.

File: example-consul-config.yaml

```yaml
global:
  name: consul
  datacenter: dc1
server:
  replicas: 3
  securityContext:
    runAsNonRoot: false
    runAsUser: 0
ui:
  enabled: true
connectInject:
  enabled: true
  default: true
controller:
  enabled: true
```

Take a look at HashiCorp's Consul and Kubernetes Deployment Guide for another example configuration, and refer to HashiCorp's Helm Chart Configuration guide for details on available parameters.

5. Install Consul. The following command assumes your configuration file is named `config.yaml` and is stored in the current working directory.

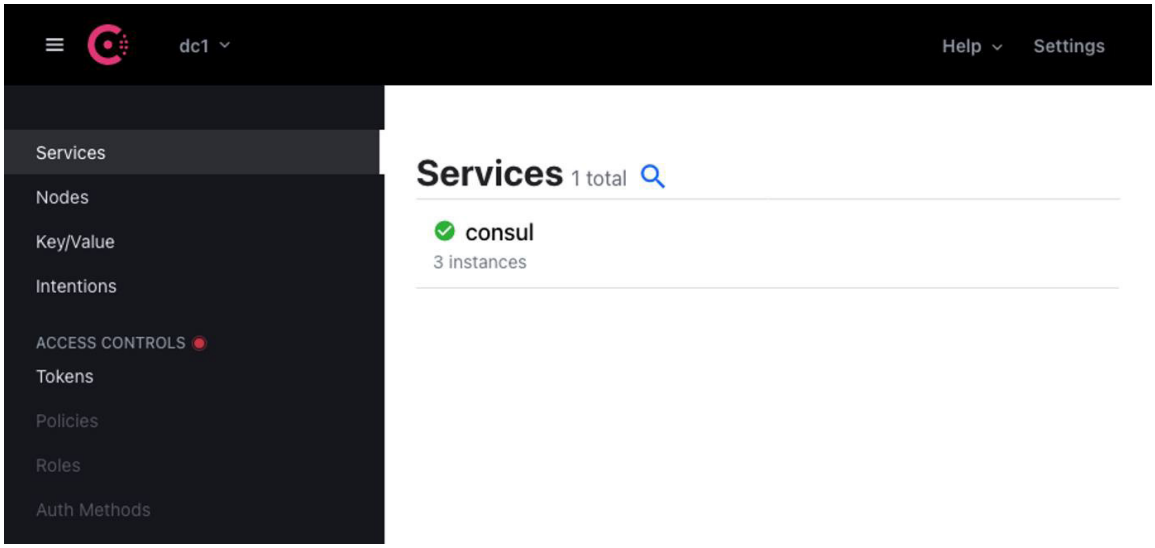
```
helm install consul hashicorp/consul --set global.name=consul -f config.yaml
```

Access the Consul Dashboard

1. Configure Kubernetes to forward the port for the Consul dashboard. The following command connects the Consul dashboard interface to your machine's port **18500**.

```
kubectl port-forward service/consul-ui 18500:80 --address 0.0.0.0
```

2. Navigate to `localhost:18500` in your browser to see the Consul dashboard.



Example Usage

Follow the steps in this section to create a couple of simple services to see the Consul service mesh in action. In this example, you use images provided by HashiCorp for some basic web services and create Kubernetes manifests for each.

Create the Service Manifests

1. Create a directory for your service manifests. Then, change into that directory. From here on, the guide assumes you are in that directory.

```
mkdir example-services
cd example-services
```

2. Create a file named example-service-backend.yaml for the first of your services. Add the contents of the example file below.

File: example-service-backend.yaml

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  name: back-end-service

---

apiVersion: v1
kind: Service
metadata:
  name: back-end-service
spec:
  selector:
    app: back-end-service
  ports:
    - port: 9091
      targetPort: 9091
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: back-end-service
  labels:
    app: back-end-service
spec:
  replicas: 1
  selector:
    matchLabels:
      app: back-end-service
  template:
    metadata:
      labels:
        app: back-end-service
      annotations:
        consul.hashicorp.com/connect-inject: 'true'
    spec:
      containers:
        - name: back-end-service
          image: nicholasjackson/fake-service:v0.7.8
          ports:
```

File: example-service-backend.yaml

```
            - containerPort: 9091
          env:
            - name: 'NAME'
              value: 'back-end-service'
            - name: 'LISTEN_ADDR'
              value: '127.0.0.1:9091'
            - name: 'MESSAGE'
              value: 'This is a response from the back-end service.'
```

The above example file creates a service and defines its deployment parameters. Take note of the `annotations` section. The `consul.hashicorp.com/connect-inject: 'true'` annotation tells Consul to inject a proxy with the service. This annotation should be included in the deployment manifest for any service you want to deploy to Kubernetes and have take part in your Consul service mesh.

3. Create another file named `example-service-frontend.yaml` for the second of your services. Add the contents of the example file below.

File: example-service-backend.yaml

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: front-end-service

---

apiVersion: v1
kind: Service
metadata:
  name: front-end-service
spec:
  selector:
    app: front-end-service
  ports:
    - port: 9090
      targetPort: 9090
---

apiVersion: apps/v1
kind: Deployment
metadata:
  name: front-end-service
  labels:
    app: front-end-service
```

File: example-service-backend.yaml

```
spec:
  replicas: 1
  selector:
    matchLabels:
      app: front-end-service
  template:
    metadata:
      labels:
        app: front-end-service
      annotations:
        consul.hashicorp.com/connect-inject: 'true'
        consul.hashicorp.com/connect-service-upstreams: 'back-end-service:9091'
    spec:
      containers:
        - name: front-end-service
          image: nicholasjackson/fake-service:v0.7.8
          ports:
            - containerPort: 9090
          env:
            - name: 'NAME'
              value: 'front-end-service'
            - name: 'LISTEN_ADDR'
              value: '0.0.0.0:9090'
            - name: 'UPSTREAM_URIS'
              value: 'http://localhost:9091'
            - name: 'MESSAGE'
              value: 'This is a message from the front-end service.'
```

This file's contents are similar to the previous file. However, it adds an additional annotation. The new annotation here — `consul.hashicorp.com/connect-service-upstreams: 'back-end-service:9091'` — tells Consul that this service has the service defined in the previous file as an upstream dependency.

Deploy the Services

1. Deploy the services to your Kubernetes cluster.

```
kubectl apply -f example-service-backend.yaml
kubectl apply -f example-service-frontend.yaml
```

2. View the pods using kubectl. Wait until the services' pods go into `Running` status before proceeding to the next step.

```
kubectl get pods
```

3. Confirm that Consul has injected proxies alongside the services.

```
kubectl get pods --selector consul.hashicorp.com/connect-inject-status=injected
```

```
NAME                                  READY   STATUS    RESTARTS   AGE
back-end-service-75cbb6cbb6-wlvf4     2/2     Running   0          3m5s
front-end-service-7dcdcc5676-zqhxh    2/2     Running   0          3m35s
```

Review the Services

1. You can see the services in action by forwarding the port for the front-end service.

```
kubectl port-forward service/front-end-service 9090:9090 --address 0.0.0.0
```

   Navigate to `localhost:9090/ui` in your browser to see the services' output.

2. Review the services in Consul's dashboard.

```
kubectl port-forward service/consul-ui 18500:80 --address 0.0.0.0
```

   Again, navigate to `localhost:18500` in your browser to see the Consul dashboard, where the new services should be listed.

Conclusion

You now have the Consul service mesh-up and running on your Kubernetes cluster. To get the most out of the service mesh, check out HashiCorp's wealth of tutorials for Consul. These can help you fine-tune the configuration to your needs and discover the myriad ways Consul can make managing your application's services easier.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Hashicorp Consul Overview
- Consul and Kubernetes Deployment Guide
- Helm Chart Configuration

Deploying Istio with Kubernetes

Istio is a service mesh, or a network of microservices, that can handle tasks such as load balancing, service-to-service authentication, monitoring, and more. It does this by deploying sidecar proxies to intercept network data, which causes minimal disruption to your current application.

The Istio platform provides its own API and feature set to help you run a distributed microservice architecture. You can deploy Istio with few to no code changes to your applications allowing you to harness its power without disrupting your development cycle. In conjunction with Kubernetes, Istio provides you with insights into your cluster leading to more control over your applications.

In this guide you will complete the following tasks:

- Create a Kubernetes Cluster and Install Helm
- Install Istio with Helm Charts
- Set up Envoy Proxies
- Install the Istio Bookinfo App
- Visualize data with Istio's Grafana addon

Important

This guide's example instructions will create several billable resources on your Linode account. If you do not want to keep using the example cluster that you create, be sure to delete it when you have finished the guide.

If you remove the resources afterward, you will only be billed for the hour(s) that the resources were present on your account. Consult the Billing and Payments guide for detailed information about how hourly billing works and for a table of plan pricing.

Before You Begin

Familiarize yourself with Kubernetes using our series A Beginner's Guide to Kubernetes and Advantages of Using Kubernetes.

Create Your Kubernetes Cluster

Important

The k8s-alpha CLI is deprecated. On **March 31st, 2020,** it will be **removed** from the linode-cli. After March 31, 2020, you will no longer be able to create or manage clusters using the k8s-alpha CLI plugin.

However, you will still be able to create and manage these clusters using Terraform. The Terraform module used is a public project officially supported by Linode, and is currently used to power the k8s-alpha CLI.

Other alternatives for creating and managing clusters include:

- The Linode Kubernetes Engine (LKE), which creates clusters managed by Linode.
- Rancher, which provides a graphical user interface for managing clusters.

There are many ways to create a Kubernetes cluster. This guide will use the Linode k8s-alpha CLI.

1. To set it up the Linode k8s-alpha CLI, see the How to Deploy Kubernetes on Linode with the k8s-alpha CLI guide and stop before the "Create a Cluster" section.

2. Now that your Linode K8s-alpha CLI is set up, You are ready to create your Kubernetes cluster. You will need **3 worker nodes** and **one master** for this guide. Create your cluster using the following command:

```
linode-cli k8s-alpha create istio-cluster --node-type g6-standard-2 --nodes 3
--master-type g6-standard-2 --region us-east --ssh-public-key $HOME/.ssh/id_rsa.pub
```

3. After the cluster is created you should see output with a similar success message:

```
Apply complete! Resources: 5 added, 0 changed, 0 destroyed.
Switched to context "istio-cluster-ka4OLOcgqHw@istio-cluster".
Your cluster has been created and your kubectl context updated.

Try the following command:
kubectl get pods --all-namespaces

Come hang out with us in #linode on the Kubernetes Slack! http://slack.k8s.io/
```

4. If you visit the Linode Cloud Manager, you will see your newly created cluster nodes on the Linodes listing page.

Install Helm

Follow the instructions in the How to Install Apps on Kubernetes with Helm guide to install Helm on your cluster. Stop before the section on "Using Helm Charts to Install Apps."

Install Istio

- For Linux or macOS users, use curl to pull the Istio project files. Even though you will use Helm charts to deploy Istio to your cluster, pulling the Istio project files will give you access to the sample `Bookinfo` application that comes bundled with this installation.

```
curl -L https://git.io/getLatestIstio | ISTIO_VERSION=1.4.2 sh -
```

- If you are using Windows, you will need to go to Istio's Github repo to find the download. There you will find the latest releases for Windows, Linux, and macOS.

> **Note**
>
> Issuing the **curl** command will create a new directory, **istio-1.4.2**, in your current working directory. Ensure you move into the directory where you'd like to store your Istio project files before issuing the **curl** command.

Install Helm Charts

1. Add the Istio Helm repo:

```
helm repo add istio.io https://storage.googleapis.com/istio-release/releases/1.4.2/charts/
```

2. Update the helm repo listing:

```
helm repo update
```

3. Verify that you have the repo:

```
helm repo list | grep istio.io
```

The output should be similar to the following:

```
istio.io        https://storage.googleapis.com/istio-release/releases/1.4.2/charts/
```

4. Install Istio's Custom Resource Definitions (CRD) with the helm chart. This command also creates a Pod namespace called istio-system which you will continue to use for the remainder of this guide.

```
helm install istio-init istio.io/istio-init
```

```
NAME: istio-init
LAST DEPLOYED: Thu Dec 12 09:20:43 2019
NAMESPACE: default
STATUS: deployed
REVISION: 1
TEST SUITE: None
```

5. Verify that all CRDs were successfully installed:

```
kubectl get crds | grep 'istio.io' | wc -l
```

You should see the following output:

```
23
```

If the number is less, you may need to wait a few moments for the resources to finish being created.

6. Install the Helm chart for Istio. There are many installation options available for Istio. For this guide, the command enables Grafana, which you will use later to visualize your cluster's data.

```
helm install istio istio.io/istio --set grafana.enabled=true
```

```
NAME: istio
LAST DEPLOYED: Thu Dec 12 09:23:02 2019
NAMESPACE: default
```

```
STATUS: deployed
REVISION: 1
TEST SUITE: None
NOTES:
Thank you for installing Istio.

Your release is named Istio.

To get started running application with Istio, execute the following steps:
1. Label namespace that application object will be deployed to by the following
command (take default namespace as an example)

$ kubectl label namespace default istio-injection=enabled
$ kubectl get namespace -L istio-injection

2. Deploy your applications

$ kubectl apply -f <your-application>.yaml

For more information on running Istio, visit:
https://istio.io/
```

7. Verify that the Istio services and Grafana are running:

```
kubectl get svc
```

The output should be similar to the following:

```
NAME                     TYPE           CLUSTER-IP      EXTERNAL-IP     PORT(S)                                                                                                                                      AGE
grafana                  ClusterIP      10.100.187.16   <none>          3000/TCP                                                                                                                                     3m35s
istio-citadel            ClusterIP      10.107.95.118   <none>          8060/TCP,15014/TCP                                                                                                                           3m35s
istio-galley             ClusterIP      10.96.238.193   <none>          443/TCP,15014/                                                                                                                               3m35s
                                                                        TCP,9901/TCP
istio-ingressgateway     LoadBalancer   10.99.127.171   104.237.148.33  15020:32094/                                                                                                                                 3m35s
                                                                        TCP,80:31380/
                                                                        TCP,443:31390/
                                                                        TCP,31400:31400/
                                                                        TCP,15029:32477/
                                                                        TCP,15030:31679/
                                                                        TCP,15031:30483/
                                                                        TCP,15032:30118/
                                                                        TCP,15443:32529/
                                                                        TCP
istio-pilot              ClusterIP      10.98.193.75    <none>          15010/TCP,15011/                                                                                                                             3m35s
                                                                        TCP,8080/
                                                                        TCP,15014/TCP
```

```
NAME                     TYPE        CLUSTER-IP       EXTERNAL-IP   PORT(S)                      AGE
istio-policy             ClusterIP   10.109.194.141   <none>        9091/TCP,15004/              3m35s
                                                                    TCP,15014/TCP
istio-sidecar-injector   ClusterIP   10.101.155.91    <none>        443/TCP,15014/TCP            3m35s
istio-telemetry          ClusterIP   10.97.162.208    <none>        9091/TCP,15004/              3m35s
                                                                    TCP,15014/
                                                                    TCP,42422/TCP
kubernetes               ClusterIP   10.96.0.1        <none>        443/TCP                      137m
prometheus               ClusterIP   10.108.217.19    <none>        9090/TCP                     3m35s
```

8. You can also see the Pods that are running by using this command:

   ```
   kubectl get pods
   ```

   The output will look similar to this:

   ```
   NAME                                    READY   STATUS      RESTARTS   AGE
   grafana-c4bcd89cb-c4cpw                 1/1     Running     0          5m13s
   istio-citadel-79945f56f7-vs8n8          1/1     Running     0          5m13s
   istio-galley-54c44fd84c-wgg2r           1/1     Running     0          5m13s
   istio-ingressgateway-bdffcd464-sw2w4    1/1     Running     0          5m13s
   istio-init-crd-10-1.4.2-pv2dc           0/1     Completed   0          7m34s
   istio-init-crd-11-1.4.2-f2pfr           0/1     Completed   0          7m34s
   istio-init-crd-14-1.4.2-hrch4           0/1     Completed   0          7m34s
   istio-pilot-6f64485fb4-k57f4            2/2     Running     2          5m14s
   istio-policy-58456b9855-jvj8s           2/2     Running     2          5m13s
   istio-sidecar-injector-b8fb8497-v89sl   1/1     Running     0          5m14s
   istio-telemetry-bb59599bd-bnzpv         2/2     Running     3          5m13s
   prometheus-fcdfd6cb5-6cjz2              1/1     Running     0          5m13s
   ```

9. Before moving on, be sure that all Pods are in the `Running` or `Completed` status.

Note

If you need to troubleshoot, you can check a specific Pod by using **`kubectl`**, remembering that you set the namespace to **`istio-system`**:

```
kubectl describe pods pod_name -n pod_namespace
```

And check the logs by using:

```
kubectl logs pod_name -n pod_namespace
```

Set up Envoy Proxies

1. Istio's service mesh runs by employing sidecar *proxies*. You will enable them by injecting them into the containers. This command is using the `default` namespace which is where you will be deploying the `Bookinfo` application.

```
kubectl label namespace default istio-injection=enabled
```

> **Note**
>
> This deployment uses automatic sidecar injection. Automatic injection can be disabled and manual injection enabled during installation via `istioctl`. If you disabled automatic injection during installation, use the following command to modify the `bookinfo.yaml` file before deploying the application:
>
> ```
> kubectl apply -f <(istioctl kube-inject -f ~/istio-1.4.2/samples/bookinfo/platform/kube/bookinfo.yaml)
> ```

2. Verify that the `ISTIO-INJECTION` was enabled for the default namespace:

```
kubectl get namespace -L istio-injection
```

```
NAME          STATUS   AGE    ISTIO-INJECTION
default       Active   141m   enabled
kube-public   Active   141m
kube-system   Active   141m
```

Install the Istio Bookinfo App

The Bookinfo app is a sample application that comes packaged with Istio. It features four microservices in four different languages that are all separate from Istio itself. The application is a simple single-page website that displays a "book store" catalog page with one book, it's details, and some reviews. The microservices are:

- `productpage` is written in Python and calls `details` and `reviews` to populate the page.
- `details` is written in Ruby and contains the book information.
- `reviews` is written in Java and contains book reviews and calls `ratings`.
- `ratings` is written in Node.js and contains book ratings. There are three versions of this microservice in the application. A different version is called each time the page is refreshed.

1. Navigate to the directory where you installed Istio.

2. The `bookinfo.yaml` file is the application manifest. It specifies all the service and deployment objects for the application. Here is just the `productpage` section of this file; feel free to browse the entire file:

File: ~/istio-1.4.2/samples/bookinfo/platform/kube/bookinfo.yaml

```yaml
...

apiVersion: v1
kind: Service
metadata:
  name: productpage
  labels:
    app: productpage
service: productpage
spec:
  ports:
  - port: 9080
    name: http
  selector:
    app: productpage
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: bookinfo-productpage
---
apiVersion: apps/v1
kind: Deployment
metadata:
  name: productpage-v1
  labels:
    app: productpage
    version: v1
spec:
  replicas: 1
  selector:
    matchLabels:
      app: productpage
      version: v1
  template:
    metadata:
      labels:
        app: productpage
        version: v1
    spec:
      serviceAccountName: bookinfo-productpage
      containers:
      - name: productpage
        image: docker.io/istio/examples-bookinfo-productpage-v1:1.15.0
```

File: ~/istio-1.4.2/samples/bookinfo/platform/kube/bookinfo.yaml

```
        imagePullPolicy: IfNotPresent
        ports:
        - containerPort: 9080
---
```

3. Start the `Bookinfo` application with the following command:

```
kubectl apply -f ~/istio-1.4.2/samples/bookinfo/platform/kube/bookinfo.yaml
```

The following output results:

```
service/details created
serviceaccount/bookinfo-details created
deployment.apps/details-v1 created
service/ratings created
serviceaccount/bookinfo-ratings created
deployment.apps/ratings-v1 created
service/reviews created
serviceaccount/bookinfo-reviews created
deployment.apps/reviews-v1 created
deployment.apps/reviews-v2 created
deployment.apps/reviews-v3 created
service/productpage created
serviceaccount/bookinfo-productpage created
deployment.apps/productpage-v1 created
```

4. Check that all the services are up and running:

```
kubectl get services
```

The output will look similar to the following:

```
NAME            TYPE           CLUSTER-IP      EXTERNAL-IP   PORT(S)              AGE
details         ClusterIP      10.101.227.234  <none>        9080/TCP             43s
grafana         ClusterIP      10.100.187.16   <none>        3000/TCP             11m
istio-citadel   ClusterIP      10.107.95.118   <none>        8060/TCP,15014/TCP   11m
istio-galley    LoadBalancer   10.96.238.193   <none>        443/TCP,15014/       11m
                                                             TCP,9901/TCP
```

```
istio-ingressgateway    LoadBalancer          10.99.127.171   104.237.148.33  15020:32094/        11m
                                                                              TCP,80:31380/
                                                                              TCP,443:31390/
                                                                              TCP,31400:31400/
                                                                              TCP,15029:32477/
                                                                              TCP,15030:31679/
                                                                              TCP,15031:30483/
                                                                              TCP,15032:30118/
                                                                              TCP,15443:32529/
                                                                              TCP
istio-pilot             ClusterIP             10.98.193.75    <none>          15010/TCP,15011/    11m
                                                                              TCP,8080/
                                                                              TCP,15014/TCP
istio-policy            ClusterIP             10.109.194.141  <none>          9091/TCP,15004/     11m
                                                                              TCP,15014/TCP
istio-sidecar-injector  ClusterIP             10.101.155.91   <none>          443/TCP,15014/TCP   11m
istio-telemetry         ClusterIP             10.97.162.208   <none>          9091/TCP,15004/     11m
                                                                              TCP,15014/
                                                                              TCP,42422/TCP
kubernetes              ClusterIP             10.96.0.1       <none>          443/TCP             11m
productpage             ClusterIP             10.110.84.77    <none>          9080/TCP            145m
prometheus              ClusterIP             10.108.217.19   <none>          9090/TCP            43s
ratings                 ClusterIP             10.110.206.217  <none>          9080/TCP            11m
reviews                 ClusterIP             10.98.21.141    <none>          9080/TCP            43s
```

5. Check that the Pods are all up:

```
kubectl get pods
```

The expected output should look similar, with all Pods running:

```
NAME                                    READY   STATUS      RESTARTS   AGE
details-v1-68fbb76fc-pz6jt              2/2     Running     0          74s
grafana-c4bcd89cb-c4cpw                 1/1     Running     0          11m
istio-citadel-79945f56f7-vs8n8          1/1     Running     0          11m
istio-galley-54c44fd84c-wgg2r           1/1     Running     0          11m
istio-ingressgateway-bdffcd464-sw2w4    1/1     Running     0          11m
istio-init-crd-10-1.4.2-pv2dc           0/1     Completed   0          14m
istio-init-crd-11-1.4.2-f2pfr           0/1     Completed   0          14m
istio-init-crd-14-1.4.2-hrch4           0/1     Completed   0          14m
istio-pilot-6f64485fb4-k57f4            2/2     Running     2          11m
istio-policy-58456b9855-jvj8s           2/2     Running     2          11m
istio-sidecar-injector-b8fb8497-v89sl   1/1     Running     0          11m
istio-telemetry-bb59599bd-bnzpv         2/2     Running     3          11m
productpage-v1-6c6c87ffff-r66dv         2/2     Running     0          74s
```

```
prometheus-fcdfd6cb5-6cjz2          1/1     Running   0          11m
ratings-v1-7bdfd65ccc-8zvk4         2/2     Running   0          74s
reviews-v1-5c5b7b9f8d-rwjgv         2/2     Running   0          74s
reviews-v2-569796655b-z5zc6         2/2     Running   0          74s
reviews-v3-844bc59d88-bwl2t         2/2     Running   0          74s
```

Note

If you do not see all Pods running right away, you may need to wait a few moments for them to complete the initialization process.

6. Check that the `Bookinfo` application is running. This command will pull the title tag and contents from the `/productpage` running on the `ratings` Pod:

```
kubectl exec -it $(kubectl get pod -l app=ratings -o jsonpath='{.items[0].metadata.name}') -c ratings -- curl productpage:9080/productpage | grep -o "<title>.*</title>"
```

The expected output will look like this:

```
&lt;title&gt;Simple Bookstore App&lt;/title&gt;
```

Open the Istio Gateway

When checking the services in the previous section, you may have noticed none had external IPs. This is because Kubernetes services are private by default. You will need to open a gateway in order to access the app from the web browser. To do this you will use an Istio Gateway.

Here are the contents of the `bookinfo-gateway.yaml` file that you will use to open the gateway:

File: ~/istio-1.4.2/samples/bookinfo/networking/bookinfo-gateway.yaml

```
1  apiVersion: networking.istio.io/v1alpha3
2  kind: Gateway
3  metadata:
4    name: bookinfo-gateway
5  spec:
6    selector:
7      istio: ingressgateway # use istio default controller
8    servers:
9    - port:
10       number: 80
11       name: http
12       protocol: HTTP
13     hosts:
14     - "*"
15 ---
```

File: ~/istio-1.4.2/samples/bookinfo/networking/bookinfo-gateway.yaml

```
16  apiVersion: networking.istio.io/v1alpha3
17  kind: VirtualService
18  metadata:
19    name: bookinfo
20  spec:
21    hosts:
22    - "*"
23    gateways:
24    - bookinfo-gateway
25    http:
26    - match:
27      - uri:
28          exact: /productpage
29      - uri:
30          prefix: /static
31      - uri:
32          exact: /login
33      - uri:
34          exact: /logout
35      - uri:
36          prefix: /api/v1/products
37      route:
38      - destination:
39          host: productpage
40          port:
41            number: 9080
```

- The `Gateway` section sets up the server and specifies the port and protocol that will be opened through the gateway. Note that the name must match Istio's named service ports standardization scheme.

- In the `Virtual Service` section, the http field defines how HTTP traffic will be routed, and the `destination` field says where requests are routed.

1. Apply the ingress gateway with the following command:

```
apply -f ~/istio-1.4.2/samples/bookinfo/networking/bookinfo-gateway.yaml
```

You should see the following output:

```
gateway.networking.istio.io/bookinfo-gateway created
virtualservice.networking.istio.io/bookinfo created
```

2. Confirm that the gateway is open:

```
kubectl get gateway
```

You should see the following output:

```
NAME               AGE
bookinfo-gateway   1m
```

3. Access your ingress gateway's external IP. This IP will correspond to the value listed under `EXTERNAL-IP`.

```
kubectl get svc istio-ingressgateway
```

The output should resemble the following. In the example, the external IP is `192.0.2.0`. You will need this IP address in the next section to access your Bookinfo app.

```
istio-ingressgateway    LoadBalancer          10.99.127.171   104.237.148.33  15020:32094/        52m
                                                                              TCP,80:31380/
                                                                              TCP,443:31390/
                                                                              TCP,31400:31400/
                                                                              TCP,15029:32477/
                                                                              TCP,15030:31679/
                                                                              TCP,15031:30483/
                                                                              TCP,15032:30118/
                                                                              TCP,15443:32529/
                                                                              TCP
```

Apply Default Destination Rules

Destination rules specify named service subsets and give them routing rules to control traffic to the different instances of your services.

1. Apply destination rules to your cluster:

```
kubectl apply -f ~/istio-1.4.2/samples/bookinfo/networking/destination-rule-all.yaml
```

The output will appear as follows:

```
destinationrule.networking.istio.io/productpage created
destinationrule.networking.istio.io/reviews created
destinationrule.networking.istio.io/ratings created
destinationrule.networking.istio.io/details created
```

2. To view all the applied rules issue the following command:

```
kubectl get destinationrules -o yaml
```

Vizualizations with Grafana

1. Open a gateway for Grafana in the same way. Create a new file called `grafana-gateway.yaml`.

File: ~/istio-1.4.2/samples/bookinfo/networking/grafana-gateway.yaml

```
apiVersion: networking.istio.io/v1alpha3
kind: Gateway
metadata:
  name: grafana-gateway
  namespace: default
spec:
  selector:
    istio: ingressgateway # use istio default controller
  servers:
  - port:
      number: 15031
      name: http-grafana
      protocol: HTTP
    hosts:
    - “*”
---
apiVersion: networking.istio.io/v1alpha3
kind: VirtualService
metadata:
  name: grafana-vs
spec:
  hosts:
  - “*”
  gateways:
  - grafana-gateway
  http:
  - match:
    - port: 15031
    route:
    - destination:
        host: grafana
        port:
          number: 3000
---
apiVersion: networking.istio.io/v1alpha3
kind: DestinationRule
metadata:
  name: grafana
spec:
  host: grafana
  trafficPolicy:
```

File: ~/istio-1.4.2/samples/bookinfo/networking/grafana-gateway.yaml

```
    tls:
      mode: DISABLE
```

2. Apply the ingress gateway with the following command:

```
kubectl apply -f ~/istio-1.4.2/samples/bookinfo/networking/grafana-gateway.yaml
```

You should see the following output:

```
gateway.networking.istio.io/grafana-gateway created
virtualservice.networking.istio.io/grafana-vs created
destinationrule.networking.istio.io/grafana created
```

3. Confirm that the gateway is open:

```
kubectl get gateway
```

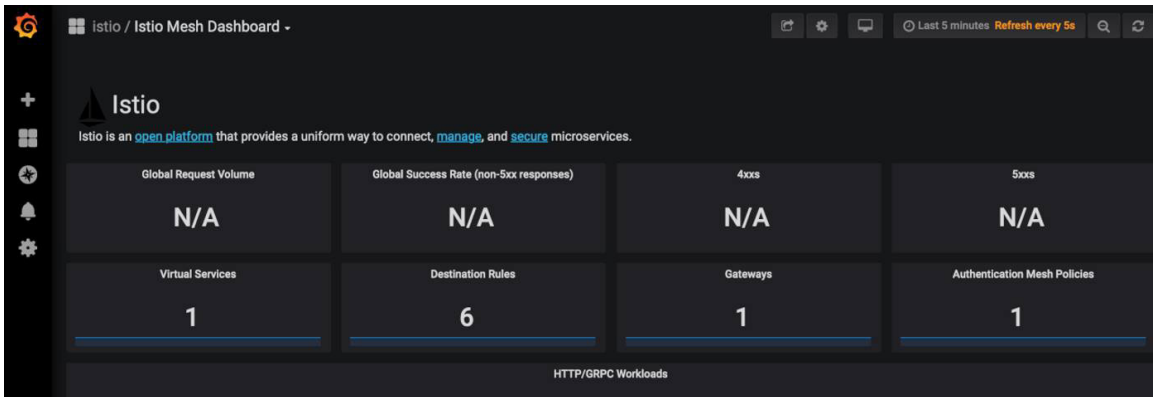
You should see the following output:

```
NAME              AGE
bookinfo-gateway  6m
grafana-gateway   48s
```

4. Once this is completed, visit the following URL in your web browser to access your *Mesh Dashboard*:

```
http://INGRESSGATEWAYIP:15031/dashboard/db/istio-mesh-dashboard
```

5. You will see the Mesh Dashboard. There will be no data available yet.

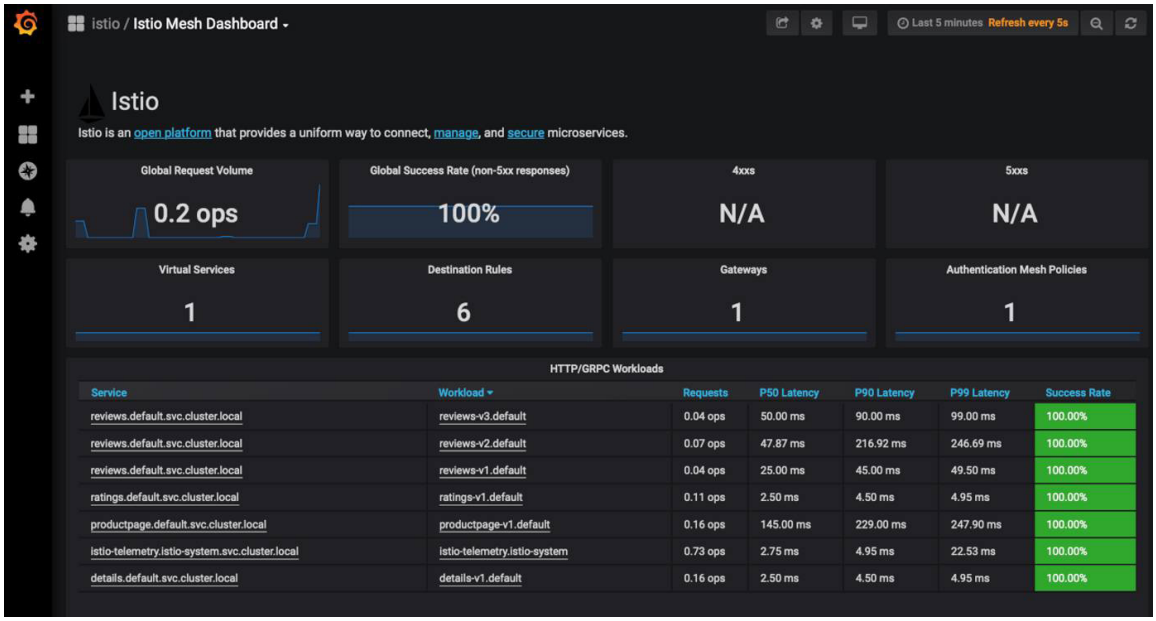


6. Send data by visiting a product page, replacing `192.0.2.0` with the value for your ingress gateway's external IP:

```
http://192.0.2.0/productpage
```

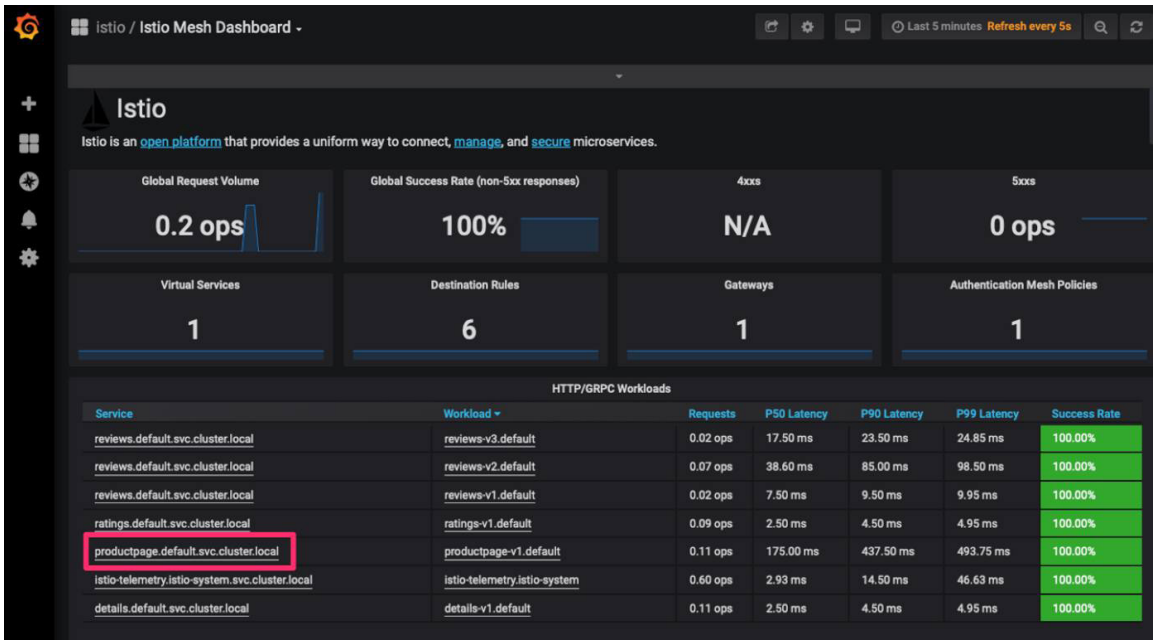
Refresh the page a few times to generate some traffic.

7. Return to the dashboard and refresh the page to see the data.

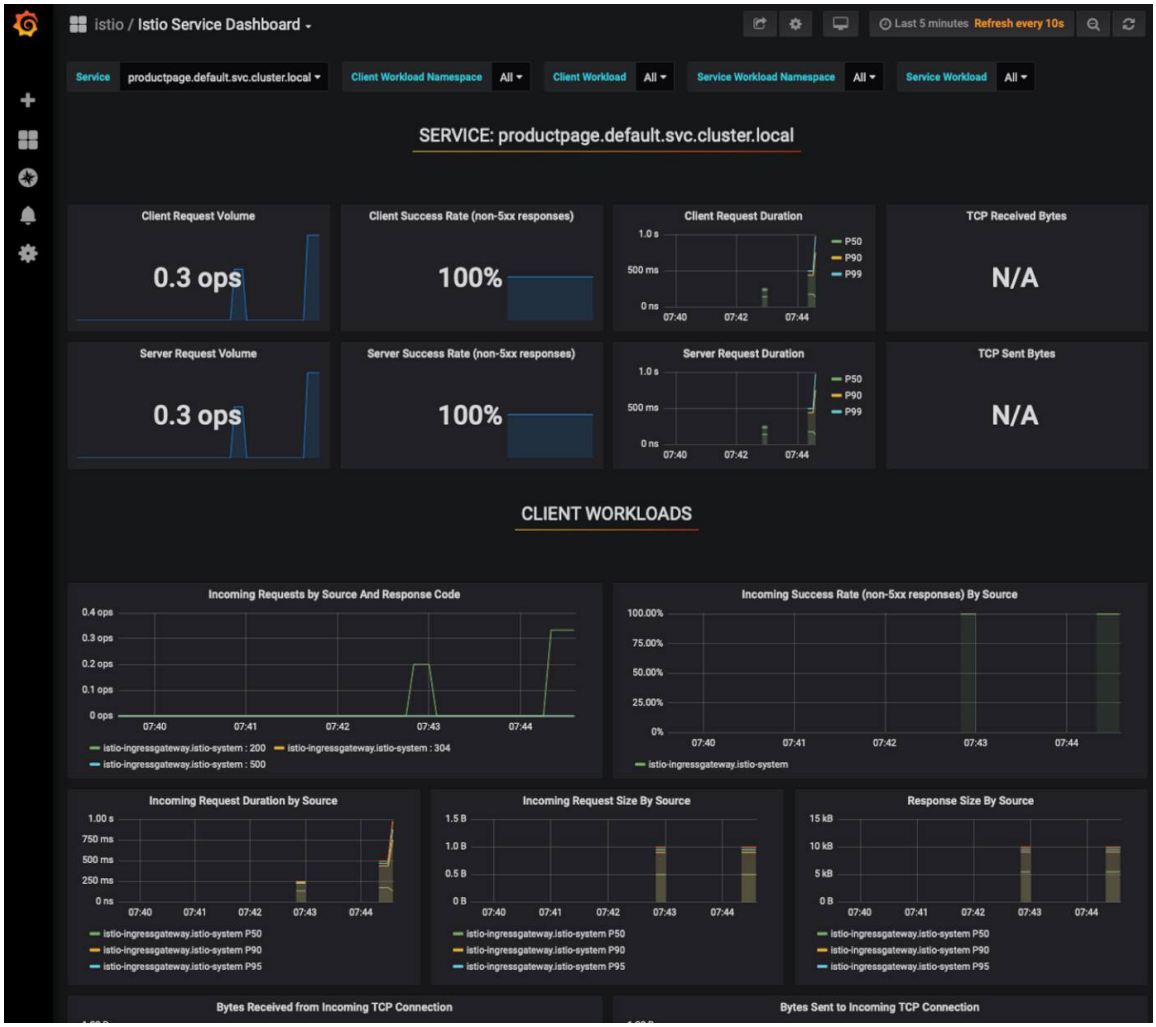


The *Mesh Dashboard* displays a general overview of Istio service mesh, the services that are running, and their workloads.

8. To view a specific service or workload you can click on them from the *HTTP/GRPC Workloads* list. Under the Service column, click `productpage.default.svc.cluster.local` from the *HTTP/GRPC Workloads* list.



9. This will open a Service dashboard specific to this service.



10. Feel free to explore the other Grafana dashboards for more metrics and data. You can access all the dashboards from the dropdown menu at the top left of the screen.

Removing Clusters and Deployments

If you at any time need to remove the resources created when following this guide, enter the following commands, confirming any prompts that appear:

```
helm uninstall istio-init
helm uninstall istio
linode-cli k8s-alpha delete istio-cluster
```

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

Istio
Istio Mesh Security
Istio Troubleshooting

Deploying Linkerd 2 with Linode Kubernetes Engine

Linkerd 2 is an ultra lightweight service mesh that monitors, reports, and encrypts connections between Kubernetes services without disturbing the existing applications. It does this by employing proxy sidecars along each instance.

Unlike Istio, another service mesh monitoring tool, it provides it's own proxies written in Rust instead of using Envoy. This makes it both lighter and more secure.

> **Note**
>
> Linkerd 1.x is still available and is being actively developed as a separate project. However, it is built on the "Twitter stack" and is not for Kubernetes. Linkerd 2 is built in Rust and Go and only supports Kubernetes.

In This Guide

This guide provides instructions to:

- Create a Kubernetes Cluster
- Install the Linkerd
- Install a Demo Application (Optional)
- Upgrade Linkerd
- Uninstall Linkerd

> **Important**
>
> This guide's example instructions create several billable resources on your Linode account. If you do not want to keep using the example cluster that you create, be sure to delete it when you have finished the guide.
>
> If you remove the resources afterward, you will only be billed for the hour(s) that the resources were present on your account. Consult the Billing and Payments guide for detailed information about how hourly billing works and for a table of plan pricing.

Before You Begin

Familiarize yourself with Kubernetes using our series A Beginner's Guide to Kubernetes and Advantages of Using Kubernetes.

Create an LKE Cluster

Follow the instructions in Deploying and Managing a Cluster with Linode Kubernetes Engine Tutorial to create and connect to an LKE cluster.

> **Note**
>
> Linkerd 2 requires Kubernetes version 1.13+. Linode Kubernetes Engine clusters currently support Kubernetes versions 1.15, 1.16, and 1.17.

Install Linkerd

Linkerd consists of the Linkerd CLI, a control plane, and a data plane. For a more detailed overview, see the Linkerd architecture.

Install the Linkerd CLI

1. To manage Linkerd you need to have the CLI installed on a local machine. The Linkerd CLI is available for Linux, macOS, and Windows on the release page.

    - For Linux, you can use the curl command for installation:

        ```
        curl -sL https://run.linkerd.io/install | sh
        ```

    - For macOS, you can use Homebrew:

        ```
        brew install linkerd
        ```

2. Verify that linkerd is installed by checking the version:

    ```
    linkerd version
    ```

3. Add Linkerd to the 'PATH' environment variable:

    ```
    export PATH=$PATH:$HOME/.linkerd2/bin
    ```

4. Use the following command to ensure that Linkerd installs correctly onto the cluster. If there are any error messages, Linkerd provides links to help you properly configure the cluster.

    ```
    linkerd check --pre
    ```

    ```
    kubernetes-api
    --------------
    √ can initialize the client
    √ can query the Kubernetes API

    kubernetes-version
    ------------------
    √ is running the minimum Kubernetes API version
    √ is running the minimum kubectl version
    ```

```
pre-kubernetes-setup
--------------------
√ control plane namespace does not already exist
√ can create non-namespaced resources
√ can create ServiceAccounts
√ can create Services
√ can create Deployments
√ can create CronJobs
√ can create ConfigMaps
√ can create Secrets
√ can read Secrets
√ no clock skew detected

pre-kubernetes-capability
-------------------------
√ has NET_ADMIN capability
√ has NET_RAW capability

linkerd-version
---------------
√ can determine the latest version
√ cli is up-to-date

Status check results are √
```

Install Linkerd Control Plane

1. Install the Linkerd control plane onto the cluster into the `linkerd` namespace:

   ```
   linkerd install | kubectl apply -f -
   ```

   This command generates a Kubernetes manifest and control plane resources. It then pipes the manifest to `kubectl apply` which instructs Kubernetes to add these resources to the cluster.

2. Validate the installation of Linkerd control plane by running the following command:

   ```
   linkerd check
   ```

   ```
   kubernetes-api
   --------------
   √ can initialize the client
   √ can query the Kubernetes API

   kubernetes-version
   ------------------
   √ is running the minimum Kubernetes API version
   ```

```
√ is running the minimum kubectl version

linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ controller pod is running
√ can initialize the client
√ can query the control plane API

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ control plane PodSecurityPolicies exist

linkerd-identity
----------------
√ certificate config is valid
√ trust roots are using supported crypto algorithm
√ trust roots are within their validity period
√ trust roots are valid for at least 60 days
√ issuer cert is using supported crypto algorithm
√ issuer cert is within its validity period
√ issuer cert is valid for at least 60 days
√ issuer cert is issued by the trust root

linkerd-api
-----------
√ control plane pods are ready
√ control plane self-check
√ [kubernetes] control plane can talk to Kubernetes
√ [prometheus] control plane can talk to Prometheus
√ tap api service is running

linkerd-version
---------------
√ can determine the latest version
```

```
√ cli is up-to-date

control-plane-version
---------------------
√ control plane is up-to-date
√ control plane and cli versions match

Status check results are √
```

3. Check the components that are installed and running:

```
kubectl -n linkerd get deploy
```

```
NAME                      READY   UP-TO-DATE   AVAILABLE   AGE
linkerd-controller        1/1     1            1           106s
linkerd-destination       1/1     1            1           106s
linkerd-grafana           1/1     1            1           104s
linkerd-identity          1/1     1            1           107s
linkerd-prometheus        1/1     1            1           105s
linkerd-proxy-injector    1/1     1            1           104s
linkerd-sp-validator      1/1     1            1           104s
linkerd-tap               1/1     1            1           103s
linkerd-web               1/1     1            1           105s
```

You can read about what each of these services do in the Linkerd architecture documentation.

The Data Plane

Each control plane component has a proxy installed in the respective Pod and therefore is also part of the data plane. This enables you to take a look at what's going on with the dashboard and other tools that Linkerd offers.

The Dashboards

Linkerd comes with two dashboards, a Linkerd dashboard and the Grafana dashboard; both are backed by metrics data gathered by Prometheus.

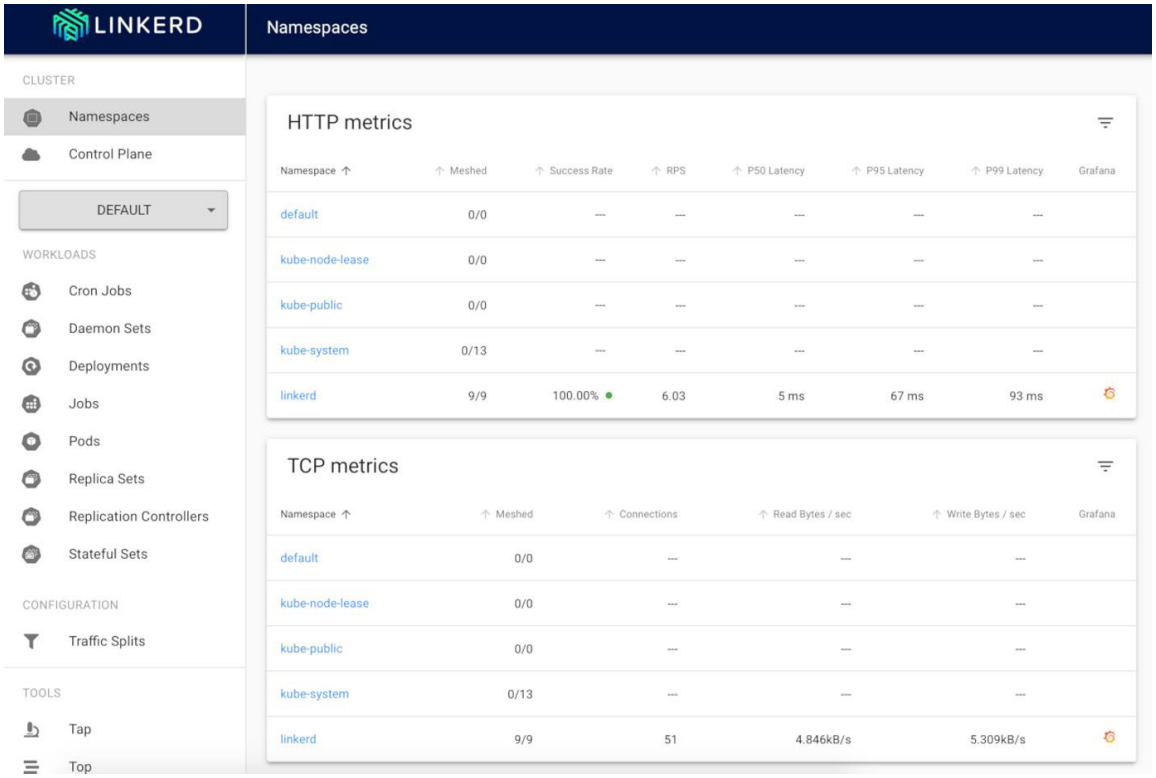
1. Start and view the Linkerd standalone dashboard that runs in the browser.

```
linkerd dasboard &
```

```
Linkerd dashboard available at:
http://localhost:50750
Grafana dashboard available at:
http://localhost:50750/grafana
Opening Linkerd dashboard in the default browser
```

- This command sets up a port forward from the linkerd-web Pod.
- If you want to expose the dashboard for others to use as well, you need to add an ingress controller.

2. The dashboard opens in the browser. If it does not, you can access it by going to http://localhost:50750:



3. The Grafana dashboard is included with Linkerd and is available at `http://localhost:50750/grafana`.

4. You can checkout the traffic that the dashboard is using by running the following command:

   ```
   linkerd -n linkerd top deploy/linkerd-web
   ```

   To see what the other Pods are doing, replace `linkerd-web` with a different Pod name, for example, to check on Grafana, use `linkerd-grafana`.

> **Note**
>
> Linkerd is not designed to be a long term metrics data store. It only stores data for 6 hours using Prometheus. However, if you can export the data using several methods.

Install Demo Application (Optional)

To demonstrate the full ease of use and utility of Linkerd, deploy Drupal on the cluster and monitor it using Linkerd.

1. Follow the How to Install Drupal with Linode Kubernetes Engine guide to install Drupal onto your LKE cluster.

Add Linkerd to Drupal

Now that Drupal is set up and running successfully on the cluster, you'll add the Linkerd service mesh to monitor the metrics.

1. Add Linkerd to the Drupal application with the following command:

   ```
   kubectl get -n default deploy -o yaml | linkerd inject - | kubectl apply -f -
   ```

   This gathers all the deployments in the `default` namespace, then pipes the manifest to `linkerd inject` which adds its proxies to the container specs and then applies it to the cluster.

   The output should look similar to this, indicating the Drupal and MySQL deployments have been *injected* with Linkerd.

   ```
   deployment “drupal” injected
   deployment “mysql” injected

   deployment.apps/drupal configured
   deployment.apps/mysql configured
   ```

2. Redeploy your Pods with the following command:

   ```
   kubectl -n default rollout restart deploy
   ```

   This redeploys the deployment manifests now that they have been injected with the Linkerd proxy sidecars. The output will look like this:

```
deployment.apps/drupal restarted
deployment.apps/mysql restarted
```

3. Issue the following command to verify that the proxies have been applied:

```
linkerd -n default check --proxy
```

```
kubernetes-api
--------------
√ can initialize the client
√ can query the Kubernetes API

kubernetes-version
------------------
√ is running the minimum Kubernetes API version
√ is running the minimum kubectl version

linkerd-existence
-----------------
√ 'linkerd-config' config map exists
√ heartbeat ServiceAccount exist
√ control plane replica sets are ready
√ no unschedulable pods
√ controller pod is running
√ can initialize the client
√ can query the control plane API

linkerd-config
--------------
√ control plane Namespace exists
√ control plane ClusterRoles exist
√ control plane ClusterRoleBindings exist
√ control plane ServiceAccounts exist
√ control plane CustomResourceDefinitions exist
√ control plane MutatingWebhookConfigurations exist
√ control plane ValidatingWebhookConfigurations exist
√ control plane PodSecurityPolicies exist

linkerd-identity
----------------
√ certificate config is valid
√ trust roots are using supported crypto algorithm
√ trust roots are within their validity period
√ trust roots are valid for at least 60 days
√ issuer cert is using supported crypto algorithm
√ issuer cert is within its validity period
```

```
√ issuer cert is valid for at least 60 days
√ issuer cert is issued by the trust root

linkerd-identity-data-plane
---------------------------
√ data plane proxies certificate match CA

linkerd-api
-----------
√ control plane pods are ready
√ control plane self-check
√ [kubernetes] control plane can talk to Kubernetes
√ [prometheus] control plane can talk to Prometheus
√ tap api service is running

linkerd-version
---------------
√ can determine the latest version
√ cli is up-to-date

linkerd-data-plane
------------------
√ data plane namespace exists
√ data plane proxies are ready
√ data plane proxy metrics are present in Prometheus
√ data plane is up-to-date
√ data plane and cli versions match

Status check results are √
```

4. You can get live traffic metrics by running the following command:

```
linkerd -n default stat deploy
```

```
NAME            MESHED       SUCCESS        RPS    LATENCY_P50    LATENCY_P95    LATENCY_P99    TCP_CONN
drupal             1/1       100.00$     0.5rps            0ms            0ms            0ms           2
mysql              1/1         0.00%     1.6rps            0ms            0ms            0ms           6
```

5. To dig deeper, try the following commands:

```
linkerd top deploy
linkerd tap deploy/drupal
```

You can also use the graphical dashboards view to show you these items in the browser.

Upgrade Linkerd

Just as with installing, upgrading Linkerd is done in multiple parts. The CLI, control plane, and data plane must all be updated separately.

Upgrade the CLI

1. Upgrade the CLI on your local machine by running the following command:

   ```
   curl -sL https://run.linkerd.io/install | sh
   ```

   You can also download the current release directly from the release page.

2. Verify the version:

   ```
   linkerd version --client
   ```

   The current version as of the writing of this guide is version `stable-2.7.1`.

Upgrade the Control Plane

1. Run the following command to upgrade the control plane:

   ```
   linkerd upgrade | kubectl apply --prune -l linkerd.io/control-plane-ns=linkerd -f -
   ```

   This will keep your current configuration and any mTLS intact.

2. Verify the upgrade with the following command:

   ```
   linkerd check
   ```

Upgrade the Data Plane

Upgrading the data plane will upgrade the proxy sidecars, auto-injecting a new version of the proxy into the Pods.

1. Use the following command to inject the new proxies, replacing namespace with the `namespace` you wish to update:

   ```
   kubectl -n namespace get deploy -l linkerd.io/control-plane-ns=linkerd -oyaml |
   linkerd inject --manual - | kubectl apply -f -
   ```

2. Issue a rollout restart of your deployment to restart the Pods, replacing `namespace` with the namespace you updated:

   ```
   kubectl -n namespace rollout restart deploy
   ```

3. Verify the upgrade with the following command:

```
linkerd check --proxy
```

4. Check the version has been updated:

```
kubectl get po --all-namespaces -o yaml | grep linkerd.io/proxy-version
```

Uninstall Linkerd

Uninstalling Linkerd is done in two steps. First, you remove the data plane proxies, then you remove the control plane. You must have cluster-wide permissions.

1. Remove the data plane proxies from your manifest files, including any injection annotations.

2. Remove the control plane with the following command:

```
linkerd install --ignore-cluster | kubectl delete -f -
```

> **Note**
>
> You may receive errors about deleting resources that haven't been created. You can safely ignore these.

3. Roll the deployments to redeploy the manifests without the Linkerd proxies, replace `namespace` with the namespace where your deployments reside:

```
kubectl -n namespace rollout restart deploy
```

When Kubernetes restarts the Pods, the Linkerd data plane will no longer be attached.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Linkerd
- Linkerd Documentation
- Linkerd Frequently Asked Questions
- Linkerd Slack

Chapter 3

Containers

Cloud-native technologies, such as containers or serverless computing, are essential for building highly portable applications in the cloud. You can design more resilient, scalable, and adaptable applications for changing environments by leveraging these technologies. These three benefits can be explained in one word: portable.

Unlike monolithic models that become cumbersome and nearly impossible to manage at larger scales, cloud-native microservices architectures are modular. This approach gives you the freedom to pick the right tool for the job, a service that does one specific function and does it well. It's here where a cloud-native approach shines, as it provides an efficient process for updating and replacing individual components without affecting the entire workload. Developing with a cloud-native mindset leads to a declarative approach to deployment: the application, the supporting software stacks, and system configurations.

Why Containers?

Think of containers as super-lightweight virtual machines designed for one particular task. Containers also are ephemeral – here one minute, gone the next. There's no persistence. Instead, persistence gets tied to block storage or other mounts within the host filesystem but not within the container itself.

Containerizing applications makes them portable! Someone can give you a container image, and you can deploy and run it across different operating systems and CPU architectures. Since containerized applications are self-contained units that get packaged with all necessary dependencies, libraries, and configuration files, code does not need to change between different cloud environments. As such, here's how containers lead to portability in a cloud-native design.

Lightweight virtualization
Containers provide an isolated environment for running applications, sharing the host OS kernel but isolating processes, file systems, and network resources.

Portable and consistent
Containers package applications and their dependencies together, ensuring they run consistently across different environments, from development to production.

Resource-efficient
Containers consume fewer resources than virtual machines, as they isolate processes and share the host OS kernel; they do not require the overhead of running a separate "guest" OS on top of the host OS.

Fast start-up and deployment
Containers start up quickly, as they do not need to boot a full OS, making them ideal for rapid deployment, scaling, and recovery scenarios.

Immutable infrastructure
Containers are designed to be immutable, meaning they do not change once built, which simplifies deployment, versioning, and rollback processes, and helps ensure consistent behavior across environments.

When Should You Consider Containers?

Containers allow you to maintain consistency. Certain aspects of development will get omitted in staging and production; for instance, verbose debug outputs. But the code that ships from development will remain intact throughout proceeding testing and deployment cycles.

Containers are very resource efficient and super lightweight. While containers are akin to virtual machines, they could be tens of megabytes as opposed to the gigs you're used to on giant (or even smaller but wastefully utilized) VMs. The lighter they get, the faster they start up, which is important for achieving elasticity and performant horizontal scale in dynamic cloud computing environments. Containers also are designed to be immutable. If something changes, you don't embed the new changes within the container; you just tear it down and create a new container. With this in mind, here are other considerations when deciding if containers should be part of your cloud-native model.

Improved deployment consistency
Containers package applications and their dependencies together, ensuring consistent behavior across different environments, simplifying deployment, and reducing the risk of configuration-related issues.

Enhanced scalability
Containers enable rapid scaling of applications by quickly spinning up new instances to handle increased demand, optimizing resource usage, and improving overall system performance.

Cost-effective resource utilization
Containers consume fewer resources than traditional virtual machines, allowing businesses to run more instances on the same hardware, leading to cost savings on cloud infrastructure.

Faster development and testing cycles
Containers facilitate a seamless transition between development, testing, and production environments, streamlining the development process and speeding up the release of new features and bug fixes.

Simplified application management
Container orchestration platforms manage the deployment, scaling, and maintenance of containerized applications, automating many operational tasks and reducing the burden on IT teams.

Container Best Practices

There are many ways to run your containers, and they're all interoperable. For instance, when migrating from AWS, you simply re-deploy your container images to the new environment and away you and your workload go. There are different tools and engines you can use to run containers but the components of your application remain independent from the platform. All of them have different resource utilization and price points. If you're hosting with Linode (Akamai's cloud computing services), you can run your containers using Linode Kubernetes Engine (LKE). You can also spin up Podman, HashiCorp Nomad, or Docker Swarm, or Compose on a virtual machine.

These open-standard tools allow you to quickly go through development and testing with the added value of simplified management when using a service like LKE. Kubernetes becomes your control plane with all the knobs and dials to orchestrate your containers with tools built on open standards. In addition, if you decide to use a platform-native offering like AWS Elastic Container Service (ECS), you'll pay for a different sort of utilization.

Another important part of containers is understanding registries, which you use to store and access your container images. One recommendation is Harbor. A CNCF project, Harbor allows you to run your private container registry, allowing you to control the security around it.

Always be testing and have a very in-depth regression test suite to ensure your code is of the highest quality for performance and security. Containers should also have a plan for failure. If a container fails, what does that retry mechanism look like? How does it get restarted? What sort of impact is that going to have? How will my application recover? Does stateful data persist on the mapped volume or bind mount?

Here are some additional best practices for using containers as part of your cloud-native development model.

Use lightweight base images
Start with a lightweight base image, such as Alpine Linux or BusyBox, to reduce the overall size of the container and minimize the attack surface.

Use container orchestration
Use container orchestration tools such as Kubernetes, HashiCorp Nomad, Docker Swarm, or Apache Mesos to manage and scale containers across multiple hosts.

Use container registries
Use container registries such as Docker Hub, GitHub Packages registry, GitLab Container registry, Harbor, etc., to store and access container images. This makes sharing and deploying container images easier across multiple hosts and computing environments.

Limit container privileges
Limit the privileges of containers to only those necessary for their intended purpose. Deploy rootless containers where possible to reduce the risk of exploitation if a container is compromised.

Implement resource constraints
Set resource constraints such as CPU and memory limits to prevent containers from using too many resources and affecting the system's overall performance.

Keep containers up-to-date
Keep container images up-to-date with the latest security patches and updates to minimize the risk of vulnerabilities.

Test containers thoroughly
Before deploying them to production, ensure that they work as expected and are free of vulnerabilities. Automate testing at every stage with CI pipelines to reduce human error.

Implement container backup and recovery
Implement a backup and recovery strategy for persistent data that containers interact with to ensure that workloads can quickly recover in case of a failure or disaster.

How Cloud Containers Work And Their Benefits

How Cloud Containers Work And Their Benefits

Tech trends come and go but cloud containers are one tech that's here to stay. Their origins can be traced back to 1982 Unix, but containers didn't gain wide acceptance until the last decade as the next logical step from virtualization. Today they are a popular means of application modernization and deployment.

The goal of containerization – that is, the process of migrating legacy apps to containers – is to offer a better way to create, package, and deploy complex software applications across different environments. Containerization provides a way to make applications less complex to deploy, update, change, modify, and scale.

Containers are increasingly popular in cloud environments because of their light weight relative to virtual machines (VMs). Many organizations view containers as an alternative to VMs' large-scale workloads.

What Are Cloud Containers?

Compute containers contain application code along with its libraries and function dependencies so they can be run anywhere, whether on a desktop PC, traditional IT server infrastructure, or the cloud.

They are small, fast, and portable because unlike a virtual machine, containers do not need to include a full-blown OS in every instance. All they need are the libraries and dependencies necessary to run the app, and leverage the other features and required resources from the host OS.

Containers are created from container images, which are templates that contain the system, applications, and environment of the container. With container images, much of the work of creating a container is already done for you. All you have to do is add the compute logic. There are many different templates for creating use-specific containers, just as there are libraries and templates for developing code.

There are multiple container template sites but the market leader is Docker, which kicked off the container trend in 2013. Docker is a set of tools that allows users to create container images, push or pull images from external registries, and run and manage containers in many different environments. It also runs the largest distribution hub of container templates. To learn how to install Docker on your Linux system, see our Installing and Using Docker guide.

Containers are significantly reduced in size and complexity, and often perform only a single function. Just because they are small doesn't mean they don't have to be managed. Containers are maintained through a process known as orchestration, which automates much of the operational tasks needed to run containerized workloads and services.

Orchestration covers managing a container's lifecycle, provisioning, deployment, scaling up or down, networking, load balancing, and more. There are several orchestration apps, but far and away the most popular is Kubernetes originally designed by Google and now maintained by the Cloud Native Computing Foundation.

Containers vs. Virtual Machines

Containers are regularly compared to VMs here and elsewhere, and for good reason. They operate on the same concept, which is the operation of multiple application environments on the same physical hardware.

VMs are considered the foundation of the first generation of cloud computing. With the advent of 64-bit computing, servers evolved past the 4GB memory limit of 32-bit processors. The arrival of multi-core produced processing power for multiple virtual environments. With enough memory and cores it is possible to run a hundred or more VMs on one physical system.

A VM needs a full operating environment that consumes one to two gigabytes of memory, regardless of whether it's on Windows Server or a version of Linux. A container is a significantly reduced operating environment and uses as little as 6MB of memory.

The benefit is you can have hundreds of containers on one robust server, so long as you have the memory and processing power to handle it all.

VM hypervisors virtualize the physical hardware, and containers virtualize the operating system. The hypervisor manages and coordinates all I/O and machine activity, balances out the load, and processes all physical tasks such as processing and data movement.

A container manager like Kubernetes handles software tasks that the container is not set up for. The app within the container has what it needs with its libraries and dependencies. If it needs something else from the OS, the container manager handles it.

It is not an either/or decision when it comes to VMs and containers. They can co-exist easily, with containers inside VMs working away.

How Do Cloud Containers Work?

Container technology was born with the first separation of partitions and chroot processes in Unix, which was later added to Linux. Containers bundle their dependency files and libraries in the container rather than rely on the underlying OS. The apps that run in containers are not full-blown, complex apps that run in a standard virtual or non-virtual environment. Each container operates in virtual isolation with each application accessing a shared OS kernel without the need for VMs.

Cloud containers are designed to virtualize a single application, whether it's a simple single-purpose app or a MySQL database. Containers have an isolation boundary at the application level rather than at the server level so the container is isolated if there is a problem. If there was an app crash or unexplained excessive consumption of resources by a process, it only affects that individual container and not the whole VM, or whole server. The orchestrator is able to spin up another container to replace the problematic container. It also shuts down and restarts the container with the issue.

The Benefits of Containers in Cloud Computing

The benefits of using containers are numerous. First, the use of templates is similar to how classes and libraries work in object-oriented programming (OOP). In OOP, you create a class or object and then reuse it in multiple apps. The same holds true for containers. A single container image is used to create multiple containers. The OOP concept of inheritance also applies to containers since container images act as the parent for other, more customized container images.

Containers run consistently on a desktop, local server, or the cloud. This makes testing them before deployment uncomplicated. Some scenarios require a test bed similar in scale to the deployment setting, which means dedicating considerable resources for the test environment. Containers can be tested locally before cloud deployment with the knowledge that performance will be consistent.

The primary advantage of containers, especially when compared to a VM, is that containers are lightweight and portable. Containers share the machine OS kernel, which eliminates a lot of overhead. Their smaller size compared to VMs means they can spin up quickly and better support cloud-native applications that scale horizontally. Other advantages include:

1. Platform independent: Containers carry all their dependencies with them, and you can use them on different Linux flavors so long as you don't make kernel calls.

2. Support modern development architectures: Due to a combination of their deployment portability/consistency across platforms and their small size, containers are an ideal fit for modern development and application methodologies, such as Agile, DevOps, serverless, and microservices.

3. Improve performance: Containerized apps are typically big apps broken down into manageable pieces. This has multiple benefits, not the least of which is performance improvements because if a component needs increased resources, the container automatically scales to offer more CPU cores, memory, or networking, then scales down when the load drops.

4. Efficient debugging: Another benefit of containerization over monolithic apps is it becomes quicker to find performance bottlenecks. With a monolithic app, developers have to do a lot of trial and error and process of elimination to find a performance bottleneck. When broken into components, the offending code becomes more visible and the developers can zoom in on the problem spot faster.

5. Hybrid/multi-cloud support: Because of their portability, containers can migrate back and forth between on-prem and the cloud. They can also move from one cloud provider to another.

6. Application modernization: A common way to modernize a legacy on-prem application is to containerize it and move it "as is" to the cloud. This model is known as "lift and shift," and is not recommended. On-prem apps behave differently than cloud-native apps and just moving an on-prem app to the cloud unchanged doesn't take advantage of cloud benefits like automatic scaling up and down.

7. Improve utilization: Under a monolithic app, the whole app and all its memory use has to increase performance. This slows down the server. With a containerized app, just that performance-intensive component needs to scale. It does it automatically, and the orchestrator scales up resources when required, and then scales down when the task is done.

Conclusion

Containers are an increasingly popular way for companies to migrate on-premises apps to the cloud and reap all the benefits the cloud brings: scale, elasticity, DevOps development, and off-loading on-prem resources to a cloud provider.

The technology is mature, with a number of competitors to Docker, including Microsoft Azure, and competitors to Kubernetes, such as Red Hat OpenShift. Most cloud providers offer some ready-made container and orchestration services, including we here at Linode, with a managed Kubernetes service.

Open Standard Monitoring Alternative to Amazon CloudWatch

Copy and paste-able command line examples appear in the technical documentation here:

- Prometheus

Prometheus & Grafana

Open-source metrics and monitoring for real-time insights.
Free industry-standard monitoring tools that work better together. Prometheus is a powerful monitoring software tool that collects metrics from configurable data points at given intervals, evaluates rule expressions, and can trigger alerts if some condition is observed. Use Grafana to create visuals, monitor, store, and share metrics with your team to keep tabs on your infrastructure.

Deploying the Prometheus & Grafana Marketplace Apps

The Linode Marketplace allows you to easily deploy software on a Linode using the Linode Cloud Manager.

1. Log in to the Cloud Manager and select the **Marketplace** link from the left navigation menu. This displays the Linode Compute **Create** page with the **Marketplace** tab pre-selected.

2. Under the **Select App** section, select the app you would like to deploy.

3. Fill out all required **Options** for the selected app as well as any desired **Advanced Options** (which are optional). See the Configuration Options section for details.

4. Complete the rest of the form as discussed within the Getting Started > Create a Linode.

5. Click the **Create Linode** button. Once the Linode has provisioned and has fully powered on, **wait for the software installation to complete**. If the Linode is powered off or restarted before this time, the software installation will likely fail. To determine if the installation has completed, open the Linode's Lish console and wait for the system login prompt to appear.

6. Follow the instructions within the Getting Started After Deployment section.

Software installation should complete within 5-10 minutes after the Linode has finished provisioning.

Configuration Options

Prometheus & Grafana Options

Here are the additional options available for this Marketplace App:

Field	Description
Admin Email for the Server	This email is require to generate the SSL certificates. *Required*

Field	Description
Your Linode API Token	Your Linode `API Token` is needed to create DNS records. If this is provided along with the `subdomain` and `domain` fields, the installation attempts to create DNS records via the Linode API. If you don't have a token, but you want the installation to create DNS records, you must create one before continuing.
Subdomain	The subdomain you wish the installer to create a DNS record for during setup. The suggestion given is `www`. The subdomain should only be provided if you also provide a `domain` and `API Token`.
Domain	The domain name where you wish to host your Wazuh instance. The installer creates a DNS record for this domain during setup if you provide this field along with your `API Token`.
The limited sudo user to be created for the Linode	This is the limited user account to be created for the Linode. This account has sudo user privileges.
The password for the limited sudo user	Set a password for the limited sudo user. The password must meet the complexity strength validation requirements for a strong password. This password can be used to perform any action on your server, similar to root, so make it long, complex, and unique.
The SSH Public Key that will be used to access the Linode	If you wish to access SSH via Public Key (recommended) rather than by password, enter the public key here.
Disable root access over SSH?	Select `Yes` to block the root account from logging into the server via SSH. Select `No` to allow the root account to login via SSH.

General Options

For advice on filling out the remaining options on the **Create a Linode** form, see Getting Started > Create a Linode. That said, some options may be limited or recommended based on this Marketplace App:

- **Supported distributions:** Ubuntu 20.04 LTS
- **Recommended plan:** All plan types and sizes can be used.

Getting Started after Deployment

Access Grafana and Prometheus

To access the front end interfaces for either Grafana or Prometheus, first obtain the credentials. Then, open your web browser and navigate to the *Location* URL of the app you wish to access. In the login prompt that appears, enter the username and password as shown in the *credentials.txt* file.

Obtain the Credentials

1. Log in to your new Compute Instance using one of the methods below:

    - **Lish Console:** Within the Cloud Manager, navigate to **Linodes** from the left menu, select the Compute Instance you just deployed, and click the **Launch LISH Console** button. Log in as the `root` user. See Using the Lish Console.

    - **SSH:** Log in to your Compute Instance over SSH using the `root` user. See Connecting to a Remote Server Over SSH for assistance.

2. Once logged in, run the following command:

    ```
    cat /root/credentials.txt
    ```

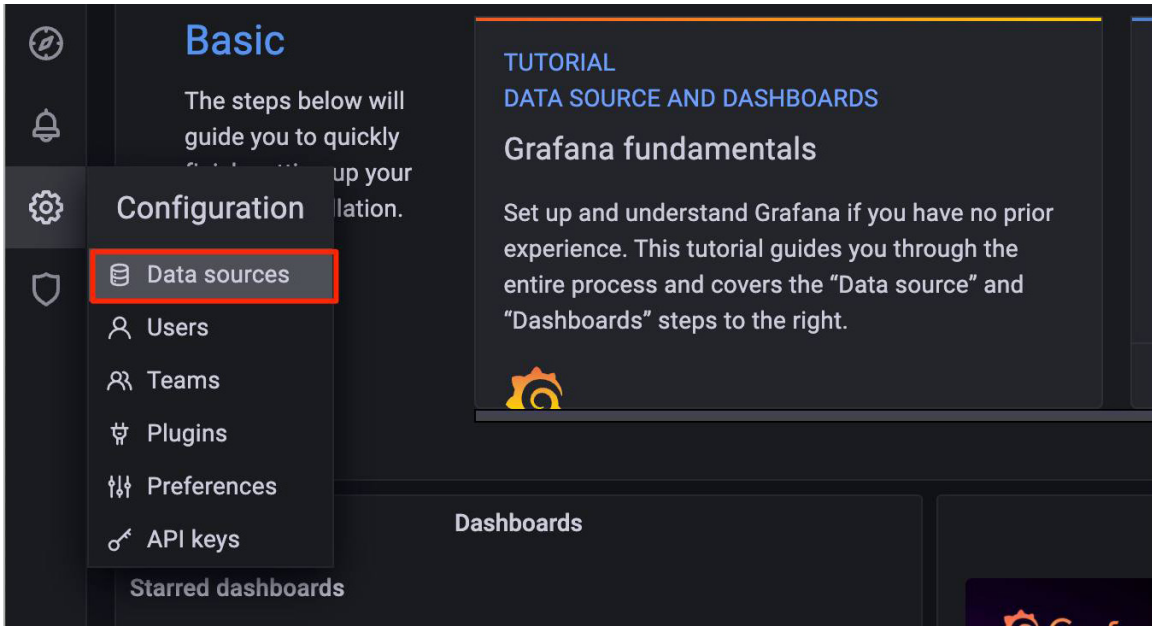
3. This displays the credentials and endpoint URL for both Prometheus and Grafana, as shown in the example output below.

    ```
    ##################
    # Prometheus #
    ##################
    Location: https://192-0-2-1.ip.linodeusercontent.com/prometheus
    Username: prometheus
    Password: htyjuykuyhjyrkit648648#$#%^GDGHDHTTNJMYJTJ__gr9jpoijrpo ##############
    # Grafana #
    ###############
    Location: https://192-0-2-1.ip.linodeusercontent.com
    Username: admin
    Password: ewtghwethetrh554y35636#$_0noiuhr09h)
    ```

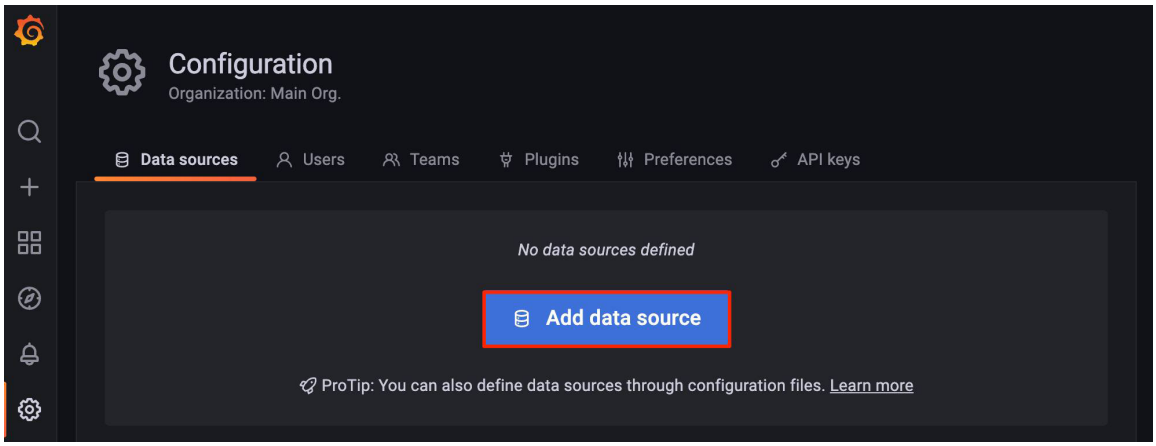
Add Prometheus as a Data Source to Grafana

1. Log in to the Grafana frontend. See Access Grafana and Prometheus.

2. On the main menu, hover over the gear icon to open the *Configuration* menu. Then click **Data Sources**.

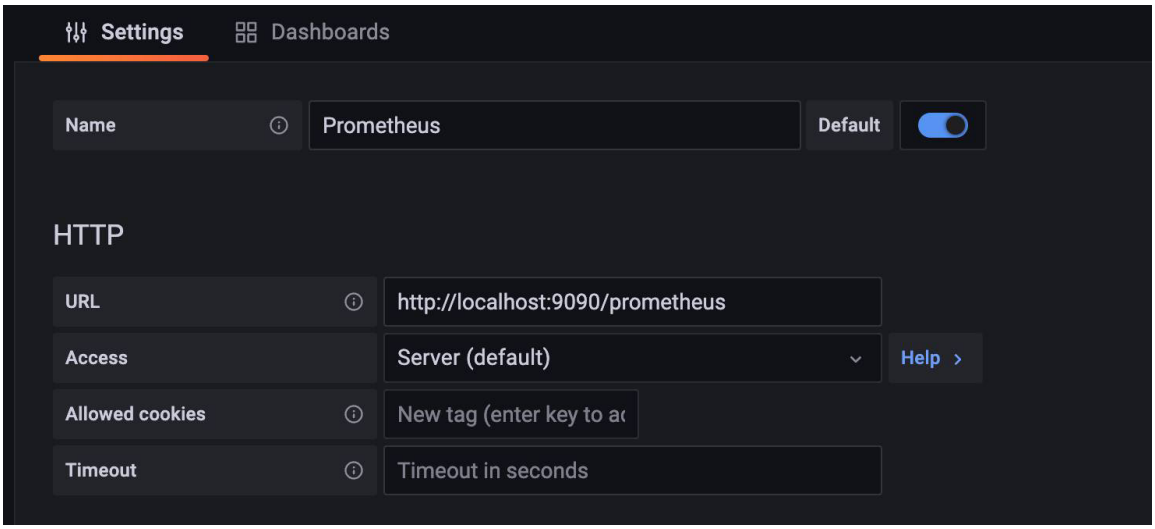


3. Within the *Data* sources page that appears, click the **Add data source** button.



4. Select Prometheus from the *Time* series database section of the *Add data source* page.

5. A data source labeled *Prometheus* is automatically created and its configuration settings are now visible. Within the **URL** field, enter `http://localhost:9090/prometheus`. The rest of the settings can be adjusted as needed.



Now that the Prometheus Data Source is set, you can browse the available Grafana dashboards to see which dashboard fits your needs. Review the official Prometheus and Grafana documentation to learn how to further utilize your instance.

The Prometheus & Grafana Marketplace App was built by Linode. For support regarding app deployment, contact Linode Support via the information listed in the sidebar. For support regarding the tool or software itself, visit Prometheus Community or Grafana Community.

Open Standard Container Registry Alternative to Amazon ECR

Deploy Harbor through the Linode Marketplace

Harbor is an open source container registry platform, cloud-native content storage, and signing/scanning tool. Harbor enhances the open source Docker distribution by providing features like security, identification, and management. The image transfer efficiency can be improved by having a registry closer to the build and run environment. Harbor includes comprehensive security features like user administration, access control, and activity auditing, as well as image replication between registries.

Harbor is an excellent compliment to the Linode Kubernetes Engine (LKE). However, you cannot install Harbor on existing or new LKE clusters.

Deploying a Marketplace App

The Linode Marketplace allows you to easily deploy software on a Compute Instance using the Cloud Manager. See Get Started with Marketplace Apps for complete steps.

1. Log in to the Cloud Manager and select the **Marketplace** link from the left navigation menu. This displays the Linode **Create** page with the **Marketplace** tab pre-selected.

2. Under the **Select App** section, select the app you would like to deploy.

3. Complete the form by following the steps and advice within the Creating a Compute Instance guide. Depending on the Marketplace App you selected, there may be additional configuration options available. See the **Configuration Options** section below for compatible distributions, recommended plans, and any additional configuration options available for this Marketplace App.

4. Click the **Create Linode** button. Once the Compute Instance has been provisioned and has fully powered on, **wait for the software installation to complete**. If the instance is powered off or restarted before this time, the software installation will likely fail.

To verify that the app has been fully installed, see Get Started with Marketplace Apps > Verify Installation. Once installed, follow the instructions within the Getting Started After Deployment section to access the application and start using it.

> **Note**
>
> Estimated deployment time: Harbor should be fully installed within 2-5 minutes after the Compute Instance has finished provisioning.

Configuration Options

- **Supported distributions:** Ubuntu 22.04 LTS, Debian 11
- **Recommended plan:** All plan types and sizes can be used.

Harbor Options

- **Admin Password** *(required)*: The Harbor admin password.
- **Database Password** *(required)*: The Harbor database password.
- **Email address** *(required)*: Enter the email address to use for generating the SSL certificates.

Custom Domain (Optional)
If you wish to automatically configure a custom domain, you first need to configure your domain to use Linode's name servers. This is typically accomplished directly through your registrar. See Use Linode's Name Servers with Your Domain. Once that is finished, you can fill out the following fields for the Marketplace App:

- **Linode API Token:** If you wish to use the Linode's DNS Manager to manage DNS records for your custom domain, create a Linode API *Personal Access Token* on your account with Read/Write access to *Domains*. If this is provided along with the subdomain and domain fields (outlined below), the installation attempts to create DNS records via the Linode API. See Get an API Access Token. If you do not provide this field, you need to manually configure your DNS records through your DNS provider and point them to the IP address of the new instance.
- **Subdomain:** The subdomain you wish to use, such as www for `www.example.com`.
- **Domain:** The domain name you wish to use, such as *example.com*.

Limited User (Optional)
You can optionally fill out the following fields to automatically create a limited user for your new Compute Instance. This is recommended for most deployments as an additional security measure. This account will be assigned to the *sudo* group, which provides elevated permission when running commands with the `sudo` prefix.

- **Limited sudo user:** Enter your preferred username for the limited user.
- **Password for the limited user:** Enter a strong password for the new user.
- **SSH public key for the limited user:** If you wish to login as the limited user through public key authentication (without entering a password), enter your public key here. See Creating an SSH Key Pair and Configuring Public Key Authentication on a Server for instructions on generating a key pair.
- **Disable root access over SSH:** To block the root user from logging in over SSH, select Yes (recommended). You can still switch to the root user once logged in and you can also log in as root through Lish.

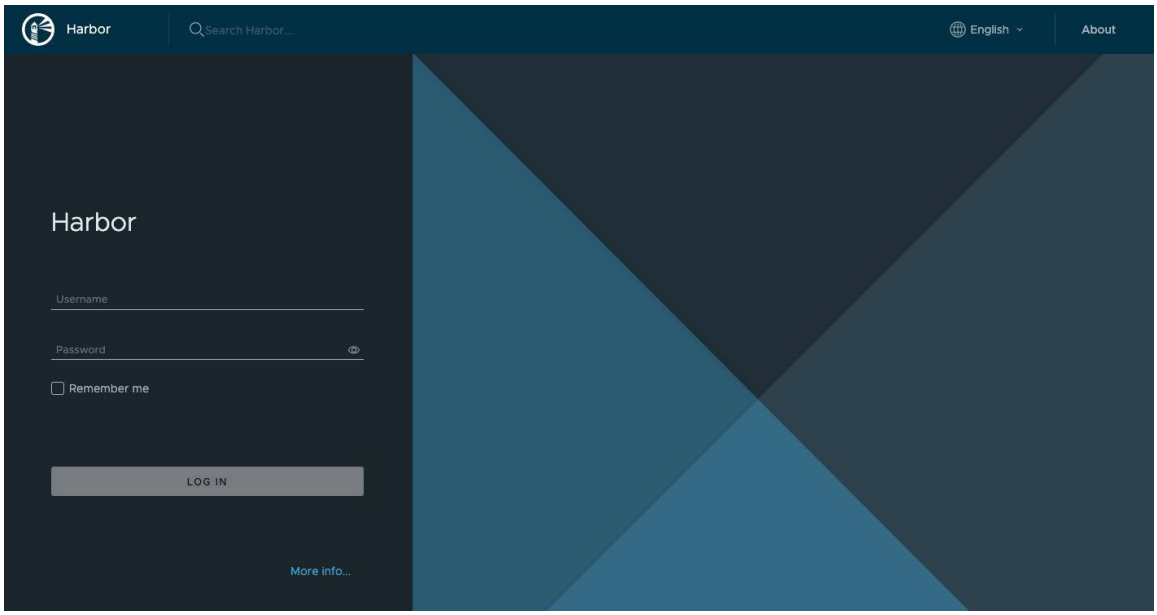
> **Warning**
>
> Do not use a double quotation mark character (“) within any of the App-specific configuration fields, including user and database password fields. This special character may cause issues during deployment.

Getting Started after Deployment

Accessing the Harbor App

1. Open a browser and navigate to the domain you created in the beginning of your deployment. You can also use your Compute Instance's rDNS, which may look like `203-0-113-0.ip.linodeusercontent.com`. See the Managing IP Addresses guide for information on viewing and setting the rDNS value.

2. In the Harbor login screen that appears, enter `admin` as the username and use the *Admin password* you created in the beginning of your deployment.



Now that you've accessed your dashboard, check out the official Harbor documentation to learn how to further utilize your Harbor instance.

> **Note**
>
> Currently, Linode does not manage software and systems updates for Marketplace Apps. It is up to the user to perform routine maintenance on software deployed in this fashion.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Harbor

Chapter 4

Serverless & EDA

Event-driven architecture (EDA) is reactive to events or messages and it triggers specific actions rather than relying on direct, synchronous communication. EDA is asynchronous, which allows components to operate independently, improving system responsiveness and performance under variable workloads.

Consider two simple examples: file uploads and new user registration. Both of these operations can happen via synchronous, request-response flow (i.e., REST API), but a new request would need to be made for a status update on the file upload or to trigger the next action to be taken after the new user data gets inserted into the database. Imagine you have a bunch of task runners continually polling for messages; they work tirelessly through periods of radio silence or unrelated chatter to occasionally get a message they can act on. You can see where this isn't the most efficient use of the elasticity of on-demand cloud computing resources. EDA resolves this matter with a push-based approach.

Event-driven systems can quickly scale by adding or removing components as needed and can be highly resilient to failures, as the system can continue functioning even if one component is unavailable. EDA also is well-suited for real-time processing and handling large volumes of data, as components can react to events and process data as it arrives without waiting for a complete dataset.

Why Should You Consider EDA?

Enhanced system flexibility
The loosely coupled nature of an event-driven architecture allows you to easily modify, add, or remove components without affecting the entire system, making it adaptable to changing requirements.

Improved scalability
EDA supports easy horizontal scaling, allowing businesses to handle increased workloads or traffic by adding more instances of components or services as needed.

Increased system resiliency
EDA's asynchronous communication and decoupled components contribute to improved fault tolerance, as the failure of one component does not necessarily cause a system-wide outage.

Real-time processing capabilities
EDA enables real-time processing of large data volumes and complex event patterns, making it suitable for businesses that require immediate insights or responses to rapidly changing conditions.

Optimized resource usage
By reacting to events only when they occur, EDA helps optimize resource utilization and reduces the need for continuously running processes, potentially leading to cost savings and improved efficiency.

Serverless Computing

EDA enables application development models like serverless computing, allowing code to be portable and cloud-agnostic so you can choose your provider based on features, supported language, costs, and other requirements. Functions-as-a-Service (FaaS) is a popular product offered by many cloud providers, which allows users to manage functions and application infrastructure all in one. Responsibility of managing the underlying infrastructure to the cloud provider, including server provisioning, scaling, and maintenance, allowing developers to focus on writing code.

Familiar FaaS services like AWS Lambda, Azure Functions, and Google Cloud Functions are what are referred to as platform-native or platform-centric. They often lock you into using a specific cloud provider with proprietary features with no easy way to migrate away without redesigning layers of your deployment. Knative, an open source project designed to use Kubernetes to build container-based applications, can be used along with a service mesh to run functions on different cloud providers or self-hosted environments.

Knative can be designed to scale your application from 0 to N number of replicas within a few seconds. Scaling to 0 is fantastic because it allows Kubernetes and Knative to reallocate resources as needed. Your one snippet of code can scale resources automatically because it can be invoked several times in parallel. At their core, the platform-native FaaS offerings mentioned earlier aren't favorable because of unpredictable pricing. By running Knative on our compute instances via our managed Kubernetes service, you pay one flat and predictable price and don't have to worry about pay-per-execution pricing that kicks in after some free tiers.

Why Should You Consider Serverless?

Cost efficiency
Serverless computing pay-as-you-go pricing model can lead to cost savings, as businesses only pay for the compute time they use without allocating resources in advance.

Improved scalability
Serverless computing can automatically scale resources to match demand, ensuring applications can handle increased workloads without manual intervention or downtime.

Reduced operational overhead
With serverless computing, the cloud provider manages the underlying infrastructure, freeing IT teams to focus on application development, innovation, and other strategic initiatives.

Faster time-to-market
The simplified development and deployment processes offered by serverless computing can help businesses accelerate the release of new features, updates, and bug fixes, enhancing their competitive advantage.

Flexibility and adaptability
Serverless computing allows businesses to build and deploy applications using a variety of programming languages and technologies, making it easier to adapt to changing requirements or incorporate new technologies as needed.

As mentioned earlier, serverless computing is based on event-driven architecture, meaning that functions get triggered by events such as HTTP requests, file uploads, database updates, and so on. This can help to simplify the application architecture and improve scalability.

Serverless functions also should be stateless. They don't store any data or state between invocations, ensuring that functions are easily scalable and you can replace them if they fail. They also should be short-lived, ensuring that resources don't get wasted and the function can scale quickly. If a function's task is long-running, evaluate whether a constantly running service is a better fit.

Don't forget to also monitor and log your serverless functions to ensure they're performing as expected and identify any issues or errors. Use tools like log aggregators and application performance monitoring (APM) tools like Prometheus and Grafana. And don't forget to secure your functions using best practices such as authentication, authorization, and encryption. This ensures that the application is secure and that sensitive data is protected. Test them thoroughly before deploying them to production to ensure that they work as expected and are free of vulnerabilities.

Serverless computing can be cost-effective, but it's important to use cost-optimization techniques such as function optimization, resource sharing, and auto-scaling to reduce costs and improve efficiency. Evaluate your workload, usage patterns, and requirements to determine whether serverless computing is cost-effective for your particular use case. Consider expected usage patterns, performance requirements, and the pricing structure of the serverless platform you choose to use.

Break Down Your Code: An Introduction to Serverless Functions and FaaS

Break Down Your Code: An Introduction to Serverless Functions and FaaS

Functions, serverless, and Kubernetes — when you're preparing to build your first containerized application, these tools and their underlying concepts can blur together. In this post, we'll demystify these essential topics for building scalable cloud-native applications.

What is Kubernetes?

Kubernetes is a container orchestration used to manage the lifecycle of containers and the nodes they run on.

Breakdown:

- In general, a **container** is an application packaged with its dependencies that do not rely on the underlying OS for additional libraries.
- Containers are grouped into **Pods** that run on **nodes**.
- A **node** is a unit of compute, often a VM running Linux (but technically any virtual or physical server in your cluster running your Pods or groups of containers).

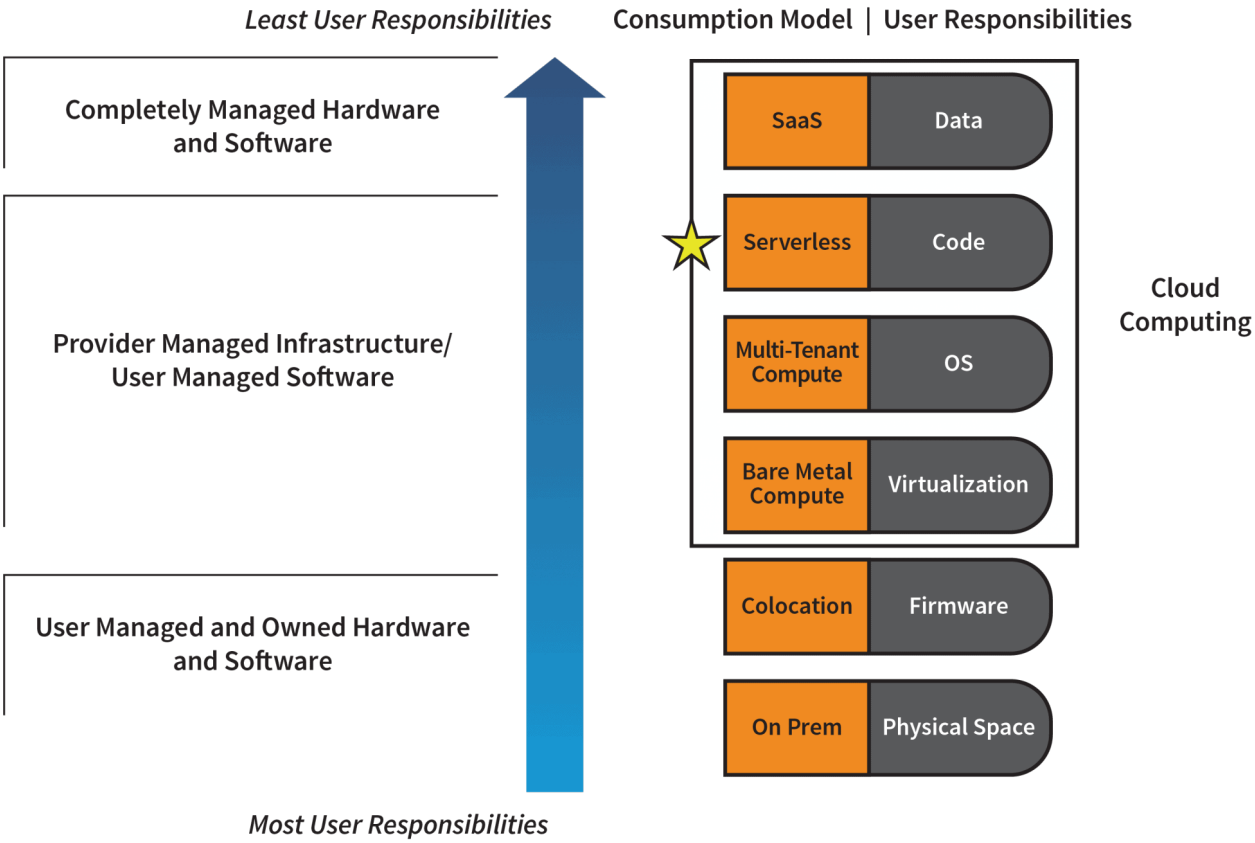
Kubernetes will **orchestrate** the creation of nodes and deploy, destroy, and move containers and pods around to nodes based on criteria defined by the developer.

Kubernetes is crucial to building and automating containerized applications for scale and high availability. If you're new to Kubernetes, we have lots of educational content to help you get started.

Kubernetes is very different from serverless functions and FaaS but is often used to power the backend that makes these architectures feasible.

What is Serverless?

Serverless is a development model that enables developers to focus on shipping code via containers without the need to manage servers or other cloud infrastructure. The cloud provider is responsible for managing everything from the server/OS level and down to provide a highly streamlined environment for developers to write and deploy code.

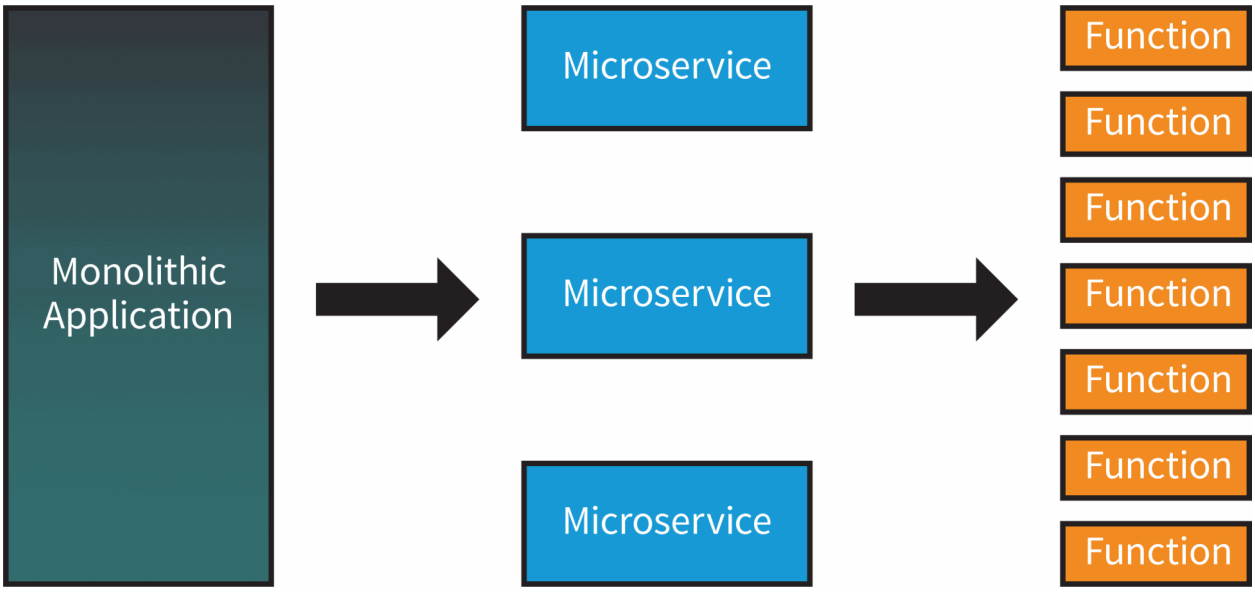


Terminology may vary between different providers, but serverless architecture shifts the responsibility from the developer to the cloud provider for everything from the server level and down.

What is a Function?

Building out modular components for an application, typically referred to as microservices, enables developers to segment “chunks” of code for functionalities that are frequently used. These chunks are known as functions, which are executed when triggered by an event.

Any user activity that triggers an event or a series of events can be deployed as a function. For instance, a user signing up on your website may trigger a database change, which may, in turn, trigger a welcome email. A chain of serverless functions can handle the backend work.



What is Functions as a Service (FaaS)?

The “as a Service” moniker is used for almost everything in our industry today. Generally speaking, ***anything*** as a Service means a cloud provider handles all of the backend infrastructure to provide streamlined access to a feature. FaaS enables developers to build and run functions with no infrastructure to maintain. FaaS offerings typically charge by execution time, meaning you’re only billed while a slice of code is running. This can be very cost-effective compared to running a server 24/7.

FaaS platforms make functions easy to deploy and manage by building them on top of a specific infrastructure technology like Kubernetes or providing a GUI to create functions and connect them to applications without writing any code. (Learn more about low-code and “no-code” applications.)

Community-submitted functions make it easier to find and implement logic to optimize application performance. FaaS is a popular product offered by many cloud providers, which allows users to manage functions and application infrastructure all in one.

Interested in Functions on Akamai?
If you currently use functions from any provider as part of your application development, we want to hear from you. Akamai’s cloud computing services are exploring adding functions to our cloud platform. Take the survey to let us know what you’re looking for, or sign up to be contacted when a functions service is available in beta.

How to Architect a Serverless Application

How to Architect a Serverless Application

Code, a set of instructions to a computer packaged as an application, only serves a purpose when there is a computer (server) to interact with. A serverless application is not an application that seemingly runs without any hardware. It's an architecture based on functions that work similarly to microservices. It utilizes modern programming strategies that employ a group of functions triggered in response to events, rather than a monolithic application, to optimize server performance. In this case, developers can disregard hardware and focus exclusively on creating the necessary functions to create an application. This guide explains how all of this works. In many cases, serverless applications require less work and produce better results, at a lower cost than other solutions.

What Is Serverless?

Applications cannot operate independently without proper support. Typically, developers work with a team consisting of a DBA, DevOps, or a system administrator to manage an application. However, with serverless architecture, developers focus only on the code for their application and do not worry about the server or hardware. Serverless applications provide functionality like automatic scaling, provisioning, built-in service integration, automated configuration, and high availability without any additional effort from the developer. The developer's only concern is the code used to create the application. This kind of application significantly reduces costs by automatically scaling both up and down to handle the current load with minimal human interaction.

A serverless application can support traditional desktop applications, back-end services, and serverless web applications. In comparison to microservices, serverless applications represent a method of running an application, while microservices represent a method of designing an application. Unlike microservices, serverless applications don't run continuously, require an event to begin execution, and individual functions perform only one task. Microservice can run continuously and support multiple tasks or functions. The primary advantage a serverless application over a microservice is that it activates when an event occurs and stops once the task is complete. The costs of running a serverless application are therefore less than a microservice in situations where an application is expected to receive frequent usage spikes.

A robust application environment can be created by combining serverless applications and microservices. Each technology has its own particular use, and relying on each when appropriate can result in a flexible and cost-effective solution.

What Are Serverless Applications Used For?

Due to the low startup costs and ability to handle lightweight applications, serverless applications are ideal for startups creating mobile and web application. Here are some other common use cases:

- Situations where traffic tends to be unpredictable.

- Internet of Things (IoT) applications, because both IoT and serverless applications are event based.
- Applications that see frequent and significant changes, as serverless applications combine well with Continuous Integration and Continuous Delivery (CI/CD).
- Applications that can be broken down into individual functions, and then combined to create a Packaged Business Capability (PBC).

How Is Building a Serverless App Different Than a Typical App?

Developing serverless applications requires a slightly different process than a monolithic application or microservice, partly because you're dependent on the hosting service. Developers need to understand the hosting service's Application Programming Interface (API) to create the application and configure each event and function accordingly. Because serverless applications are so dependent on a particular hosting service, they come with some risk of vendor lock-in.

Testing and debugging serverless applications requires special attention. Ironically, the problem comes from the very technology that saves time, effort, and money in other ways. Because functions only execute when triggered by an event, intermittent errors can be difficult to find without a thorough testing and debugging strategy. Additionally, connectivity issues between serverless applications, configuration problems, or other factors can make it difficult to track down the root cause of a problem.

Performance is also a crucial consideration when designing a serverless application. Depending on the hosting service, the application can be cached for a specific period of time after it stops running. This enables a quick startup if another event triggers the function before the cache is cleared. However, if the cache is cleared, there could be a delay while the server reloads the application. Even with the best planning, performance can be uneven.

A serverless application requires an event to trigger it, in the form of a message. In this case, the problem is that the message may contain special legal or other handling requirements, which makes sending the information problematic. These messaging issues can extend into transactional requirements because built-in latency often makes transactions hard to track. Proper logging is essential to ensure that a transaction has actually occurred; however, this can also slow down the application.

Considering the Serverless Application Process

There's a process to follow when architecting a serverless application, whether the resulting software represents back-end services, front-end services, or both. This process differs from working with monolithic applications, microservices, Packaged Business Capabilities (PBCs), or other software development patterns. The idea is to break a software requirement down into smaller pieces until it's possible to describe a each individual piece very simply.

1. Define individual services that perform specific tasks.
2. Define individual functions that perform one and only one task, to make up the services.
   Build a collection of functions that define each element of a service in detail, and in the most basic way possible. It should not be possible to break a task down any further than the function level.
   While lambda functions are most common, any language works if the service provider supports it.
3. Define events that trigger the functions. Remember that serverless applications work on the premise that a function starts, performs a task, and then stops.
4. Create configuration files that describe each function, including function name, script name, function environment, resources required for execution, and at least one event that causes the function to run. Optionally, include the packaging used to bundle the function and resources together in a single, easily installed file.
5. Create a configuration provider file that describes how the function interacts with the framework supporting the serverless application. This file should describe the framework environment and indicate the stage of the application, such as "development" or "production."
6. Create a service configuration file that details the provider file, function files, and any plugins required for the service. *Plugins* are specialized software that extend the functionality provided by the framework environment, scripting solution, or other elements that make up the service. The service configuration file can also contain details about authentication, authorization, and environmental concerns that affect the service as a whole.

Using Serverless Applications Versus Microservices

When architecting a solution that includes serverless applications, it's important to have an understanding of different approaches to working with code. It's essential to know the strengths and weaknesses of various solution models and determine if a combination of models results in the best implementation.

What Are Microservices?

Keep the differences between microservices and serverless applications in mind as you consider architecting a solution based on one or the other, or both. As previously mentioned, microservices are essentially a method of designing an application, rather than a method of deciding how to run the application. Microservices are often employed in these use cases:

- Applications that require scalability.
- Big applications that manage large amounts of data in various ways.
- Migration of legacy applications from a monolithic architecture to a microservice architecture.
- Situations in which an organization supports multiple applications and needs to use components from one application in another.

When considering a microservice architecture, there are certain advantages to consider, such as:

- Scalability, as each microservice is independent and can be scaled separately using techniques such as data partitioning and multiple instances to solve performance problems.
- Reliability, because if one microservice goes down, it's easy to use another instead.
- Platform independence, as microservices can connect to different platforms.
- Ease of experimentation, since different scenarios can be tried without bringing the entire application down.
- Team autonomy, because each microservice is its own codebase and has its own data store that doesn't rely on other microservices.

However, microservices do present some drawbacks in comparison to serverless applications, including:

- High startup costs due to the need to carefully architect connectivity between microservices.
- Difficulty in testing the entire solution, although testing individual microservices is easier.
- Complex debugging processes because the source of a problem can't be determined until all logs are examined.
- Security issues because microservices are prone to misconfiguration.

What Are Back-End Services?

Back-end services are responsible for making an application function. Back-end services typically include load balancers, database managers, business logic services, and services that perform Create, Read, Update, and Delete (CRUD) operations on data. Back-end services also include message queues for storing requests and event managers. The latter are especially important for serverless applications because events trigger the functions.

It's essential to understand a back-end service since it lacks a user interface and it doesn't interact directly with the user. Depending on the service provided, a serverless application can provide perfect support for back-end services, as a front-end service (e.g. user interface element) can make a request to the back end, which performs the task, then stops until another event occurs.

What Are Front-End Services?

Front-end services handle the user interface, date presentation and validation, and other aspects that focus on the user experience. A front-end service may also provide a query API, such as a REST API. This allows third-party applications to interact with back-end services without a user interface. Additionally, a front-end service manages various facets of an application, such as obtaining credentials to be authenticated by the back-end services. The back end also tells the front end what a particular actor is authorized to do. Serverless applications are well-suited for certain elements of front-end services because they typically spend significant time waiting for user input. Using a serverless application reduces costs significantly because there are no expenses for inactivity. When the user is ready to interact with the application, clicking a button creates an event that triggers a function in the serverless application. Here, payment for processing time is measured in milliseconds rather than minutes.

Conclusion

Serverless applications offer developers a cost-effective solution for quick application development without worrying about hardware. Serverless applications and microservices are not mutually exclusive. In fact, it often pays to combine them in large applications to leverage the best of both technologies. It's essential to remember that serverless applications start, run, and stop, so performance often suffers as a result. Microservices, on the other hand, are designed to run for long periods of time, sacrificing low cost for higher performance.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Geeks4Geeks: Serverless Computing
- Geeks4Geeks: Why Serverless Apps?

DIY Functions: Comparing Serverless Toolsets

DIY Functions: Comparing Serverless Toolsets

FaaS is typically associated with your cloud provider of choice, which provides convenience and predictability for infrastructure management. However, as developers and businesses increasingly opt for multicloud deployments to achieve redundancy and reduce costs, this creates demand for provider-agnostic FaaS platforms that are portable across workloads and cloud providers.

After working with Justin Mitchel of Coding for Entrepreneurs to teach developers about Knative with our on-demand course, we're comparing a few of the popular, provider-agnostic, and open-source FaaS tools and frameworks.

OpenFaaS: OpenFaaS is a popular toolset for functions experimentation and testing on non-production workloads. The paid version, OpenFaaS Pro, has a GUI and is a simple way to deploy event-driven functions and microservices. OpenFaas requires a license for most workloads, and they advise against using the free Community edition in production.

Fission: Fission is a feature-rich functions framework that provides a wide range of pre-built integrations right out of the box, especially for webhooks that trigger events and send you notifications via your chosen tool. Fission caches functions to deliver better performance over time as your application uses some functions more than others.

Knative: Knative provides a set of building blocks for creating and managing serverless Kubernetes applications, including automatic scaling and event-driven computing. Knative allows you to declare a desired state for your cluster state and scale efficiently, including scaling to zero pods. Knative is highly customizable and extensible and is backed by a large open-source community.

Ultimately, all of these tools are similar in what they can accomplish. However, they differ in setup effort and how much configuration is needed to achieve goals that are specific to each application.

	Native Autoscaling	Scaling to Zero	Native Supported Languages	License Required	Deployment Type	Functions Management	Setup Effort
OpenFaas Community Edition	Requests per Second (RPS)	No	Dockerfile, Go, NodeJS, Python, Java, Ruby, PHP, C#	No	Kubernetes, Docker	GUI	Low
OpenFaas Pro	RPS, Capacity, CPU usage	Yes	Dockerfile, Go, NodeJS, Python, Java, Ruby, PHP, C#	Yes	Kubernetes, Docker	GUI	Low
Fission	CPU usage	No	Python, NodeJS, Go, C#, PHP	No	Kubernetes, Docker	Command Line	Medium
Knative	Request concurrency, Target utilization, KPA metrics	Yes	NodeJS, Java, Python, Ruby, C#, PHP, Go	No	Kubernetes	Command Line	High

Like other developer tools, there are many options on the market and more to come as functions usage and capabilities continue to expand. We're aiming to make functions seamless on the Akamai cloud platform.

Take the survey to let us know what you'd like to see in a Functions service, or sign up here and we'll contact you when Functions is available in beta.

Open Standard Alternative to AWS Lambda

Try Knative

Knative is a Kubernetes-based platform for running serverless. Serverless means you can scale your application to 0 running instances but can quickly scale up to N number of instances within a few seconds. Scaling to 0 is fantastic because it allows Kubernetes and Knative to reallocate resources as needed.

If you couple that with our managed Kubernetes autoscaling feature (which will add compute nodes to your cluster), you can have a very robust system with not much invested financially. The investment for Knative comes in the form of the learning curve to get it running and unlocking continuous delivery/deployment.

Here's what we're going to cover in this article and the course:

- Using Terraform to create our Kubernetes Cluster on Linode
- Install Knative and Istio
- Configure a Knative Service and Domain Mapping
- Install cert-manager for auto-provisioning of HTTPS certificates
- Configure an Istio Gateway for HTTP and HTTPS requests (ingress)
- Implement Knative service environment variables (both ConfigMap and Secrets)

In this series, we'll learn how to install Knative to a Kubernetes cluster and deploy a containerized application!

Justin Mitchel is a father, coder, teacher, YouTuber, best-selling Udemy instructor, and the founder of Coding for Entrepreneurs. Connect with Justin on Twitter @justinmitchel.

ACCESS FREE COURSE

Open Standard Messaging Alternative to Amazon Simple Queue Service (SQS)

Copy and paste-able command line examples appear in the technical documentation here:

- Kafka (Developed by Apache)
- Storm (Developed by Apache)

An Introduction to Apache Kafka

Apache Kafka, often known simply as Kafka, is a popular open-source platform for stream management and processing. Kafka is structured around the concept of an event. External agents, independently and asynchronously, send and receive event notifications to and from Kafka. Kafka accepts a continuous stream of events from multiple clients, stores them, and potentially forwards them to a second set of clients for further processing. It is flexible, robust, reliable, self-contained, and offers low latency along with high throughput. LinkedIn originally developed Kafka, but the Apache Software Foundation offers the current open-source iteration.

An Overview of Apache Kafka

Kafka can be thought of as a re-implementation, or an evolution, of a traditional database for a streaming world. Whereas the databases you are probably familiar with store data in tables with well-defined attributes, keys, and schemas, Kafka is much more freeform. Kafka's purpose is to receive, store, and transmit a record of real-time *events*.

In a typical workflow, one or more producer applications send key-value messages about a predefined topic to a Kafka cluster. A cluster consists of one or more servers, which are also called brokers, and each cluster typically hosts messages for many topics. One of the brokers in the cluster receives these messages and writes them to a log file corresponding to the topic. These log files are called *partitions*, and topics usually contain several partitions. Messages might also get replicated to some of the other nodes within the cluster.
Other processes known as consumers can then read and process the events in each partition. You can write these consumer and producer applications yourself or use third-party offerings.

> **Note**
>
> At the time of writing this guide, the version of Apache Kafka is release 2.7.

Advantages of Apache Kafka

1. **High Throughput:** Kafka is based on a highly optimized and efficient architecture. This allows Kafka to handle both low latency as well as high throughput.
2. **Highly Reliable:** Kafka can handle high-volume data flow from multiple producers (who write to Kafka) and to multiple consumers (who poll Kafka for data to read).
3. **Durability:** Kafka stores events in a simple log format and, hence, the data is durable and retention policies are easy to implement. You can deploy Kafka on virtual machines, bare metal, and in the cloud.
4. **Scalability:** Without any system downtime, you can easily add or upgrade nodes for extra reliability.
5. **Message Broker Capabilities:** You can organize several Kafka message brokers into a fault-tolerant cluster and replicate data between them.
6. **Consumer Friendly:** Kafka can integrate well with many programming languages including Java (the native language of Kafka), Go, C++, Python, and REST APIs which are useful for testing and prototyping.

Kafka provides applications such as *Kafka Connect* (for integrating external components) and Kafka Streams (for stream processing), as well as security and access policies. Many vendors have jumped in with third-party extensions for legacy systems, and Kafka provides many APIs for both producers and consumers to use. However, solid programming skills are required to develop a complex Kafka application.

Use Cases for Apache Kafka

Kafka can be used in a variety of settings, but it lends itself to environments with a real-time data stream. It is an ideal format for a modern microservice-based world.

- Kafka is especially practical if the data is coming from multiple sources and intended for multiple destinations. Kafka works best with data representing a sequence of discrete events in a log-ready format.
- A suitable application for Kafka might be a simple home monitoring system. Each device in the house is a producer. Each of these producers sends status updates, alarms, maintenance reminders, and customer requests to Kafka. The events aggregate into a multi-device stream, which Kafka stores for further reference. The data remains available for alarm-monitoring services and customer web portals to access later on. The monitoring service could constantly poll for new data, while customers might review their data sometime in the future.
- Kafka can also serve vastly more complicated systems, such as hotel reservation networks. The data might come from thousands of producers, including third-party systems. It could be sent to far more demanding consumers, such as marketing, e-commerce, and reservation portals. A large Kafka cluster might consist of hundreds of brokers and have tens of thousands of topics, along with complex replication and retention policies.

The *Kafka website* mentions several high-level domains where Kafka clusters might be used. Some of these areas include:

- **Message Brokers:** Due to Kafka's low latency and reliability, it excels as a buffer for inter-system communications. Its design allows for a completely modular decoupling of producers and consumers. High-speed producers, such as web applications, can immediately send events to Kafka. Consumer systems can then chew through the message buffer and carry out their more time-intensive work. Consumers can temporarily fall behind during high-volume bursts without destabilizing the system. Kafka simply continues to store the messages, and the consumers can retrieve them when they are ready.
- **Website Tracking:** Web tracking can be a very high-volume activity if a site intends to track all user actions. Kafka is a good choice for this because of its high throughput and efficient storage system. The aggregators can sweep up the data later on and rebuild the customer timeline based on the ID and topic of each event.
- **Metrics/Log Aggregation:** Kafka is suitable for logging operational data from a large number of networked devices. Kafka preserves data directly in log form and abstracts away system details, so simple end-user devices can quickly and easily interact with it.
- **Stream Processing:** Many modern web applications handle a stream of user updates and submissions. This type of data flow naturally lends itself to Kafka processing. Pipelines can evaluate, organize, and manipulate this content into a format suitable for secondary sources.

Architecture of Apache Kafka

Kafka's architecture contains the components and extensions listed below. The following sections offer a more in-depth discussion for each component and extension.

- Kafka Event Message Format
- Topics and Partitions
- Clusters and Replication (including Zookeeper)
- Producers and Consumers
- Security, Troubleshooting, and Compatibility

Kafka Event Message Format

In Kafka terminology, an event, a record, and a message all refer to the same thing. Kafka and its clients can only communicate by exchanging events. The Kafka design intentionally keeps the message format simple and concise to allow for greater flexibility. Each message contains the following fields:

- **Metadata header:** Each variable length header consists of fields including message length, key, value offsets, a CRC, the producer ID, and a magic ID (akin to a version). The Kafka APIs automatically build the headers.
- **Key:** Each application defines its own keys. Keys are opaque and are of variable length. In a typical application, the key refers to a particular user, customer, store, device, or location. It answers questions like:
  - Who or what generated this event?
  - Who or what does this event concern?

  Kafka ensures all messages with the same key are stored sequentially inside the same partition.
- **Value:** Values are opaque and are of variable length, similar to keys. The open-ended structure allows you to store any amount of data in any format. A value can be a standardized multi-field record or a simple string description, but values typically have at least some structure.
- **Timestamp:** Represents either the time the producer generated the event or the time Kafka received it.

Kafka Topics and Partitions

Each event is stored inside a Kafka *topic*, and each topic contains many events. A topic can be thought of as a file folder. Each "folder" contains many individual files representing the events. Kafka allows an unlimited number of producers to publish events on the same topic.

The events within a topic have the following capabilities :

- They are immutable and persist after being read, so they can be accessed multiple times.
- They can be stored indefinitely, subject to storage limits. By default, events are kept for seven days. Each event within a topic is stored within a designated partition.

Kafka topics can be managed via the Kafka Administration API. Kafka's topics are divided into several *partitions*. Some of Kafka's partition facts are as follow:

- Kafka groups messages together when accessing a partition, resulting in efficient linear writes. Each message obtains a sequentially increasing number and is stored at an offset within its partition. This is used to maintain strict ordering.
- Events in a partition are always read in the same order they were written, but you can choose to compact a Kafka topic.
- When compaction is set, older events do not automatically expire. However, when a new event arrives, Kafka discards older events with the same key. Only the newest update is kept. You can choose to apply a deletion policy or a compaction policy, but not both.

Kafka Clusters and Replication

Although you can certainly run Kafka on a stand-alone server, it works best when it runs on a cluster.

- Some of the servers in a cluster act as brokers to store the data. The other servers within the cluster might run services like *Kafka Connect* and *Kafka Streams* instead. Grouping the servers this way increases capacity and also allows for high reliability and fault tolerance.
- Kafka uses a central management task called *Zookeeper* to control each cluster. Zookeeper elects and tracks leaders to monitor the status of cluster members and manage the partitions.
- Zookeeper optionally maintains the *Access Control List* (ACL). You must launch Zookeeper before starting Kafka, but you do not otherwise have to actively manage it. Your cluster configuration determines how Zookeeper behaves. You can use the *Kafka Administration API* to manage cluster and replication settings.

Kafka replicates the partitions between the servers so there are several backups on other servers. A set of three or four servers is typical. One of the brokers is elected the leader on a per-partition basis. The leader receives events from the producers and sends updates to the consumers. The remaining brokers serve as followers. They query the leader for new events and store backup copies.

You can configure a Kafka cluster for different levels of reliability. Kafka can send an acknowledgment upon the first receipt of a message or when a certain number of backup servers have also made a copy. The first method is faster, but a small amount of data might be lost if the master fails. Producers can elect not to receive any acknowledgments if best-effort handling is the goal. Kafka does not automatically balance any topics, partitions, or replications. The Kafka administrator must manage these tasks.

Producers and Consumers

Producers and consumers can both use the Kafka Administration API to communicate with Kafka.

- Applications use these APIs to specify a topic and send their key-value messages to the cluster.
- Consumers use the API to request all stored data or to continuously poll for updates.

The Kafka cluster keeps track of each consumer's location within a given partition so it knows which updates it still has to send. *Kafka Connect* and *Kafka Streams* help manage the flows of information to or from Kafka.

Our guide for installing Kafka includes an example of how to use the producer and consumer APIs.

Security, Troubleshooting, and Compatibility

Kafka clusters and brokers are not secure by default, but Kafka supports many security options.

- Using SSL, you can authenticate connections between brokers and clients, as well as between the brokers and the Zookeeper.
- You can enforce authorization for reading and write operations and encrypt data in transit. Kafka supports a mix of authenticated and non-authenticated clients and a high degree of granularity to the security settings.

Kafka also provides tools to monitor performance and log metrics, along with the usual error logs. Several third-party tools are offering Kafka command centers and big-picture snapshots of an entire cluster.

When upgrading Kafka brokers, or individual clients, it is important to consider compatibility issues. Schema evolution must be properly planned out, with components upgraded in the proper order along with version mismatch handling. Kafka streams provide version processing to assist with client migration.

Kafka Connect

Kafka Connect is a framework for importing data from other systems, or exporting data to them. This allows easier integration between Kafka and traditional databases.

Some of the benefits of Kafka Connect include:

- Kafka Connect runs on its own server rather than on one of the regular Kafka brokers.
- The Kafka Connect API builds upon the Consumer and Producer APIs to implement connector functions for many legacy systems.
- Kafka Connect does not ship with any production-ready connectors, but there are many open-source and commercial utilities available. For example, you might use a *Kafka* Connect connector to quickly transfer data out of a legacy relational database and into your Kafka cluster.

Kafka Streams

Kafka Streams is a library designed for rapid stream processing. It accepts input data from Kafka, operates upon the data, and retransmits the transformed data. A Kafka Streams application can either send this data to another system or back to Kafka.

Some of the benefits of Kafka Streams are:

- Kafka Streams offers utilities to filter, map, group, aggregate, window, and join data.
- **Highly Durable:** The open-source RocksDB extension permits stateful stream processing and stores partially processed data on a local disc.
- **Stream processing:** Kafka Streams provides transactional writes, which guarantee "exactly once" processing. This means that it can execute a read-process-write cycle one time, without missing any input messages nor produces duplicate output messages.

Install Kafka

You must install Java first before installing Apache Kafka. Kafka itself is straightforward to install, initialize, and run.

- The Kafka site contains a basic tutorial.
- We also have a guide on how to Install Apache Kafka which demonstrates how to construct a simple producer and consumer and process data with Kafka Streams.

Further Reference

Apache provides extensive documentation and supporting materials for Kafka.

- The Kafka documentation web page discusses the design, implementation, and operation of Kafka, with a deep dive into common tasks.
- In-depth API information is found on the Kafka JavaDocs page. You can reference a high-level overview of each class along with an explanation of the various methods.
- You can also refer to the Kafka Streams documentation which features a demo and a fairly extensive tutorial.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Apache Kafka
- Events
- Kafka Administration API
- Kafka Documentation

Streaming Data Processing with Apache Storm

Apache Storm is a big data technology that enables software, data, and infrastructure engineers to process high-velocity, high-volume data in real time and extract useful information. Any project that involves processing high-velocity data streams in real time can benefit from it.

Zookeeper is a critical distributed systems technology that Storm depends on to function correctly.

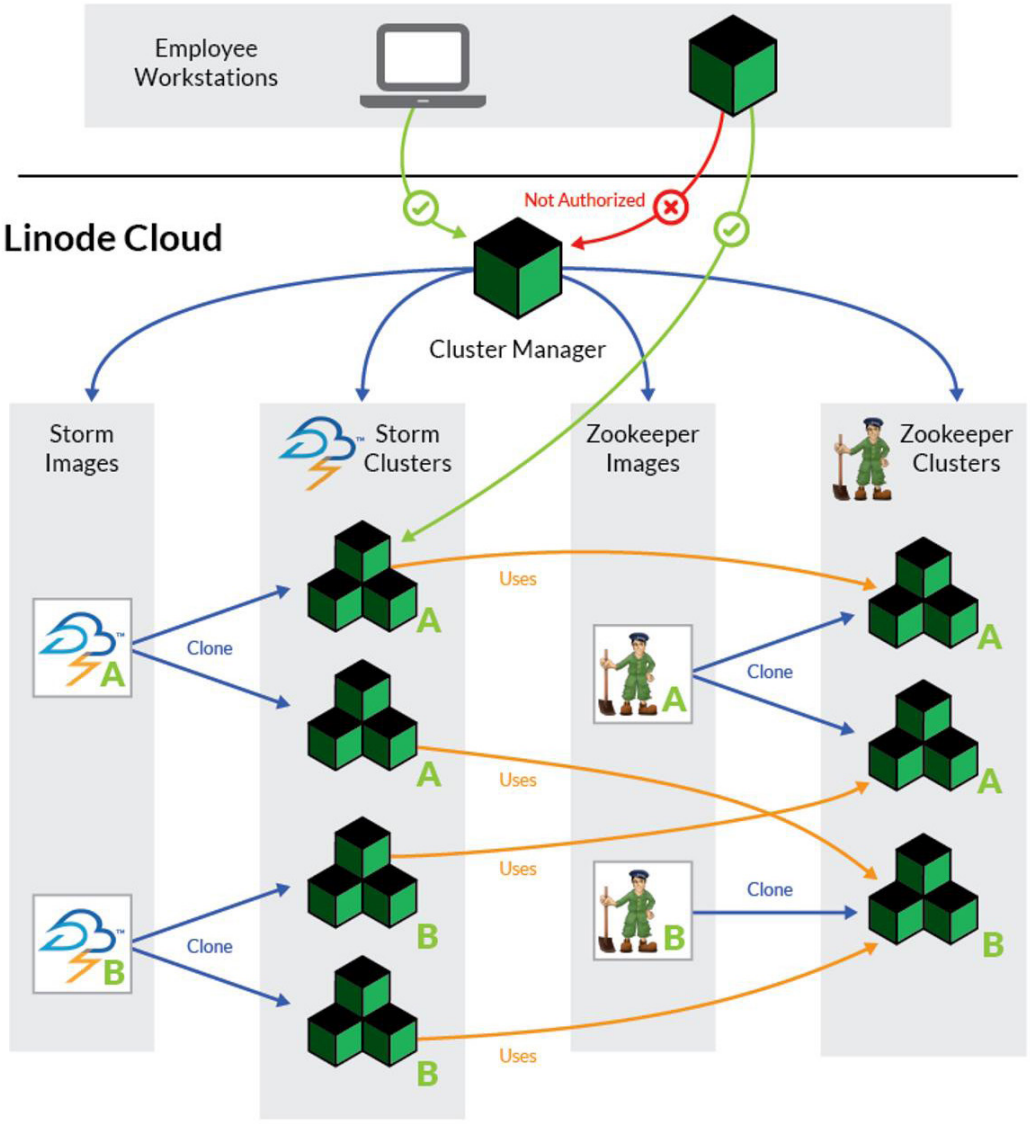
Some use cases where Storm is a good solution:

- Twitter data analytics (for example, trend prediction or sentiment analysis)
- Stock market analysis
- Analysis of server logs
- Internet of Things (IoT) sensor data processing

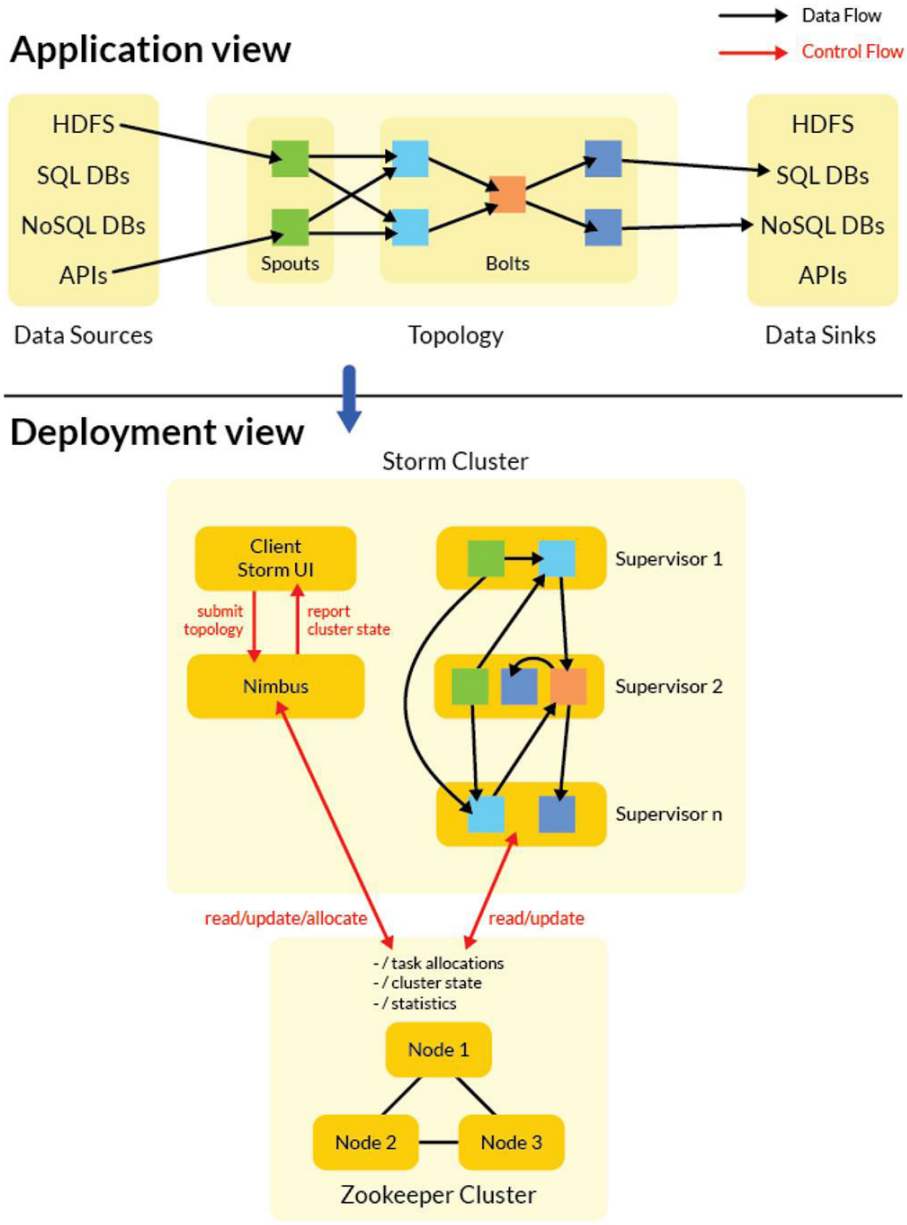
This guide explains how to create Storm clusters on the Linode cloud using a set of shell scripts that use Linode's Application Programming Interface (APIs) to programmatically create and configure large clusters. The scripts are all provided by the author of this guide via GitHub repository. This application stack could also benefit from large amounts of disk space, so consider using our Block Storage service with this setup.

> **Important**
>
> External resources are outside of our control, and can be changed and/or modified without our knowledge. Always review code from third party sites yourself before executing.

The deployed architecture will look like this:



The deployed architecture will look like this:



The application flow begins, from the client side, with the Storm client, which provides a user interface. This contacts a *Nimbus* node, which is central to the operation of the Storm cluster. The Nimbus node gets the current state of the cluster, including a list of the supervisor nodes and *topologies* from the Zookeeper cluster. The Storm cluster's supervisor nodes constantly update their states to the Zookeeper nodes, which ensure that the system remains synced.

The method by which Storm handles and processes data is called a *topology*. A topology is a network of components that perform individual operations, and is made up of *spouts*, which are sources of data, and *bolts*, which accept the incoming data and perform operations such as running functions or transformations. The data itself, called a *stream* in Storm terminology, comes in the form of unbounded sequences of tuples.

Storm (Developed by Apache)

This guide will explain how to configure a working Storm cluster and its Zookeeper nodes, but it will not provide information on how to develop custom topologies for data processing. For more information on creating and deploying Storm topologies, see the Apache Storm tutorial.

Before You Begin

OS Requirements

- This article assumes that the workstation used for the initial setup of the cluster manager Linode is running Ubuntu 14.04 LTS or Debian 8. This can be your local computer, or another Linode acting as your remote workstation. Other distributions and operating systems have not been tested.
- After the initial setup, any SSH capable workstation can be used to log in to the cluster manager Linode or cluster nodes.
- The cluster manager Linode can have either Ubuntu 14.04 LTS or Debian 8 installed.
- A Zookeeper or Storm cluster can have either Ubuntu 14.04 LTS or Debian 8 installed on its nodes. Its distribution does not need to be the same one as the one installed on the cluster manager Linode.

> **Note**
>
> The steps in this guide and in the bash scripts referenced require root privileges. Be sure to run the steps below as **root**. For more information on privileges, see our Users and Groups guide.

Naming Conventions

Throughout this guide, we will use the following names as examples that refer to the images and clusters we will be creating:

- `zk-image1` - Zookeeper image
- `zk-cluster1` - Zookeeper cluster
- `storm-image1` - Storm image
- `storm-cluster1` - Storm cluster

These are the names we'll use, but you are welcome to choose your own when creating your own images and clusters. This guide will use these names in all example commands, so be sure to substitute your own names where applicable.

Get a Linode API Key

Follow the steps in Generating an API Key and save your key securely. It will be entered into configuration files in upcoming steps.

If the key expires or is removed, remember to create a new one and update the `api_env_linode.conf` API environment configuration file on the cluster manager Linode. This will be explained further in the next section.

Set Up the Cluster Manager

The first step is setting up a central *Cluster Manager* to store details of all Storm clusters, and enable authorized users to create, manage or access those clusters. This can be a local workstation or a Linode, but in this guide will be a Linode.

1. The scripts used in this guide communicate with Linode's API using Python. On your workstation, install Git, Python 2.7 and curl:

```
sudo apt-get install python2.7 curl git
```

2. Download the project git repository:

```
git clone “https://github.com/pathbreak/storm-linode”
cd storm-linode
git checkout $(git describe $(git rev-list --tags='release*' --max-count=1))
```

3. Make the shell and Python scripts executable:

```
chmod +x *.sh *.py
```

4. Make a working copy of the API environment configuration file:

```
cp api_env_example.conf api_env_linode.conf
```

5. Open `api_env_linode.conf` in a text editor, and set `LINODE_KEY` to the API key previously created (see Get a Linode API key).

File: ~/storm-linode/api_env_linode.conf

```
1 export LINODE_KEY=fnxaZ5HMsaImTTRO8SBtg48...
```

6. Open `~/storm-linode/cluster_manager.sh` in a text editor and change the following configuration settings to customize where and how the Cluster Manager Linode is created:

    - `ROOT_PASSWORD` : This will be the root user's password on the Cluster Manager Linode and is **required** to create the node. Set this to a secure password of your choice. Linode requires the root password to contain at least 2 of these 4 character types:

        - lower case characters
        - upper case characters
        - numeric characters
        - symbolic characters

        If you have spaces in your password, make sure the entire password is enclosed in double quotes (`“`). If you have double quotes, dollar characters, or backslashes in your password, escape each of them with a backslash (`\`).

- `PLAN_ID` : The default value of `1` creates the Cluster Manager Linode as a 2GB node, the smallest plan. This is usually sufficient. However, if you want a more powerful Linode, use the following commands to see a list of all available plans and their IDs:

  ```
  cp api_env_example.conf api_env_linode.conf
  ```

  > **Note**
  >
  > You only need to run **source** on this file once in a single terminal session, unless you make changes to it.

- `DATACENTER` : This specifies the Linode data center where the Cluster Manager Linode is created. Set it to the ID of the data center that is nearest to your location, to reduce network latency. It's also recommended to create the cluster manager node in the same data center where the images and cluster nodes will be created, so that it can communicate with them using low-latency private IP addresses and reduce data transfer usage.

  To view the list of data centers and their IDs:

  ```
  source ~/storm-linode/api_env_linode.conf
   ~/storm-linode/linode_api.py datacenters table
  ```

- `DISTRIBUTION` : This is the ID of the distribution to install on the Cluster Manager Linode. This guide has been tested only on Ubuntu 14.04 or Debian 8; other distributions are not supported.

  The default value of `124` selects Ubuntu 14.04 LTS 64-bit. If you'd like to use Debian 8 instead, change this value to `140` .

  > **Note**
  >
  > The values represented in this guide are current as of publication, but are subject to change in the future. You can run **~/storm-linode/linode_api.py distributions** to see a list of all available distributions and their values in the API.

- `KERNEL` : This is the ID of the Linux kernel to install on the Cluster Manager Linode. The default value of `138` selects the latest 64-bit Linux kernel available from Linode. It is recommended not to change this setting.

- `DISABLE_SSH_PASSWORD_AUTHENTICATION` : This disables SSH password authentication and allows only key-based SSH authentication for the Cluster Manager Linode. Password authentication is considered less secure, and is hence disabled by default. To enable password authentication, you can change this value to `no` .

  > **Note**
  >
  > The options shown in this section are generated by the **linode_api.py** script, and differ slightly from the options shown using the Linode CLI tool. Do not use the Linode CLI tool to configure your Manager Node.

When you've finished making changes, save and close the editor.

7. Now, create and set up the Cluster Manager Linode:

```
./cluster_manager.sh create-linode api_env_linode.conf
```

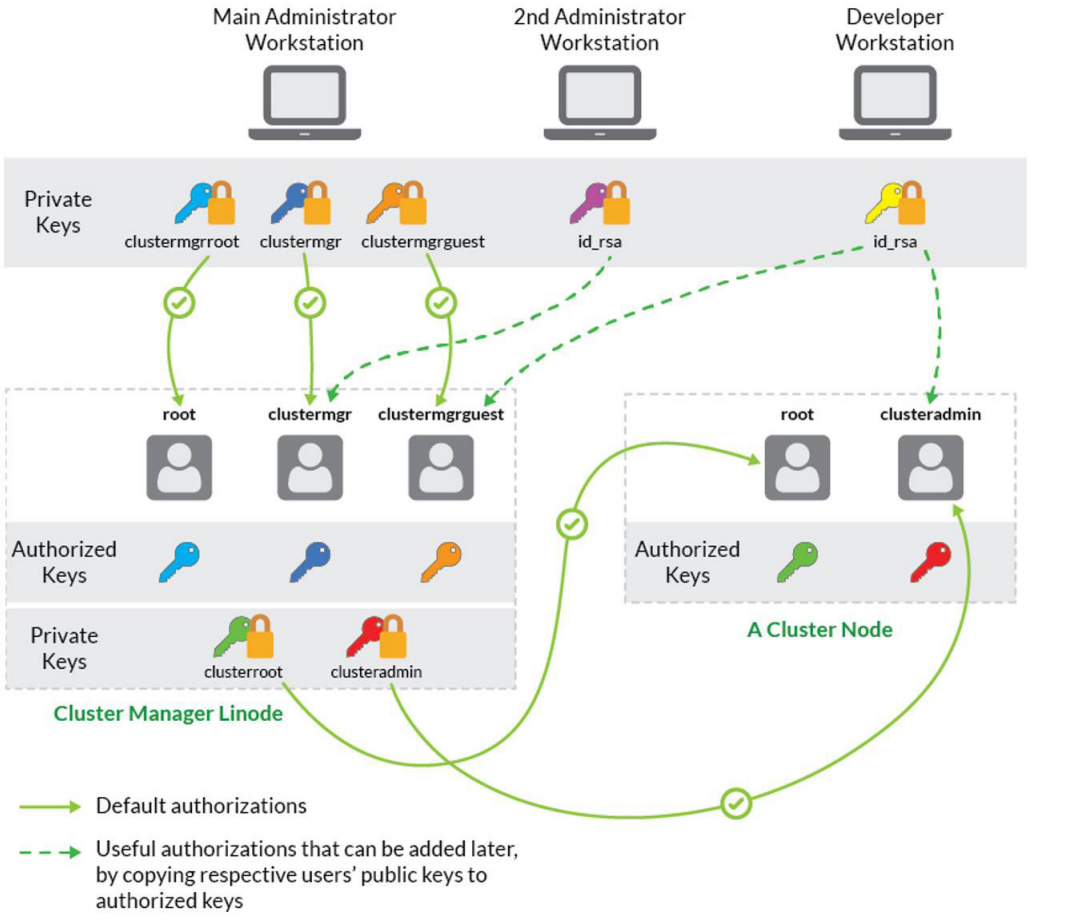
Once the node is created, you should see output like this:

```
Creating linode
Creating disk
Creating a configuration
Creating private IP for linode
Private IP address xxx.xxx.xxx.xxx created for linode 100129
Getting IP address of linode
Public IP address: xxx.xxx.xxx.xxx
Booting the linode
Cluster manager node has booted. Public IP address: xxx.xxx.xxx.xxx
```

Public IP address of cluster manager linode

Note the public IP address of the Cluster Manager Linode. You will need this when you log into the cluster manager to create or manage clusters.

8. The `cluster_manager.sh` script we ran in the previous step creates three users on the Cluster Manager Linode, and generates authentication keypairs for all of them on your workstation, as shown in this illustration:



- `~/.ssh/clustermgrroot` is the private key for Cluster Manager Linode's *root* user. Access to this user's credentials should be as restricted as possible.

- `~/.ssh/clustermgr` is the private key for the Cluster Manager Linode's *clustermgr* user. This is a privileged administrative user who can create and manage Storm or Zookeeper clusters. Access to this user's credentials should be as restricted as possible.

- `~/.ssh/clustermgrguest` is the private key for Cluster Manager Linode's *clustermgrguest* user. This is an unprivileged user for use by anybody who need information about Storm clusters, but not the ability to manage them. These are typically developers, who need to know a cluster's client node IP address to submit topologies to it.

SSH password authentication to the cluster manager is disabled by default. It is recommended to leave the default setting. However, if you want to enable password authentication for just *clustermgrguest* users for convenience, log in to the newly created cluster manager as `root` and append the following line to the end of `/etc/ssh/sshd_config`:

File: /etc/ssh/sshd_config

```
1  Match User clustermgrguest
2    PasswordAuthentication yes
```

Restart the SSH service to enable this change:

```
service ssh restart
```

> **Important**
>
> Since access to the cluster manager provides access to all Storm and Zookeeper clusters and any sensitive data they are processing, its security configuration should be considered critical, and access should be as restrictive as possible.

9. Log in to the cluster manager Linode as the `root` user, using the public IP address shown when you created it:

```
ssh -i ~/.ssh/clustermgrroot root@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
```

10. Change the hostname to something more descriptive. Here, we are changing it to *clustermgr*, but you may substitute a different name if you like:

```
sed -i -r "s/127.0.1.1.*$/127.0.1.1\tclustermgr/" /etc/hosts
echo clustermgr > /etc/hostname
hostname clustermgr
```

11. Set passwords for the *clustermgr* and *clustermgrguest* users:

```
passwd clustermgr
passwd clustermgrguest
```

Any administrator logging in as the *clustermgr* user should know this password because they will be asked to enter the password when attempting a privileged command.

12. Delete `cluster_manager.sh` from root user's directory and close the SSH session:

```
rm cluster_manager.sh
exit
```

13. Log back in to the Cluster Manager Linode – this time as *clustermgr* user – using its public IP address and the private key for *clustermgr* user:

```
ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
```

14. Navigate to your `storm-linode` directory and make a working copy of `api_env_example.conf`. In this example, we'll call it `api_env_linode.conf`:

```
cd storm-linode
cp api_env_example.conf api_env_linode.conf
```

15. Open the newly created `api_env_linode.conf` in a text editor and set `LINODE_KEY` to your API key.

    Set `CLUSTER_MANAGER_NODE_PASSWORD` to the password you set for the clustermgr user in Step 11.

File: ~/storm-linode/api_env_linode.conf

```
1  export LINODE_KEY=fnxaZ5HMsaImTTRO8SBtg48...
2  ...
3  export CLUSTER_MANAGER_NODE_PASSWORD=changeme
```

    Save your changes and close the editor.

16. The cluster manager Linode is now ready to create Apache Storm clusters. Add the public keys of anyone who will manage the clusters to `/home/clustermgr/.ssh/authorized_keys`, so that they can connect via SSH to the Cluster Manager Linode as user `clustermgr`.

Create a Storm Cluster

Creating a new Storm cluster involves four main steps, some of which are necessary only the first time and can be skipped when creating subsequent clusters.

Create a Zookeeper Image

A *Zookeeper* image is a master disk image with all necessary Zookeeper software and libraries installed. We'll create our using Linode Images The benefits of using a Zookeeper image include:

- Quick creation of a Zookeeper cluster by simply cloning it to create as many nodes as required, each a perfect copy of the image
- Distribution packages and third-party software packages are identical on all nodes, preventing version mismatch errors
- Reduced network usage, because downloads and updates are executed only once when preparing the image instead of repeating them on each node

> **Note**
>
> If a Zookeeper image already exists, this step is not mandatory. Multiple Zookeeper clusters can share the same Zookeeper image. In fact, it's a good idea to keep the number of images low because image storage is limited to 10GB.
>
> When creating an image, you should have clustermgr authorization to the Cluster Manager Linode.

1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

2. Choose a unique name for your image and create a configuration directory for the new image using the `new-image-conf` command. In this example, we'll call our new image `zk-image1` :

   ```
   ./zookeeper-cluster-linode.sh new-image-conf zk-image1
   ```

   This creates a directory named `zk-image1` containing the files that make up the image configuration:

   - **zk-image1.conf** : This is the main image configuration file, and the one you'll be modifying the most. Its properties are described in the next step. This file is named `zk-image1.conf` in our example, but if you chose a different image name, yours may vary.

   - **zoo.cfg** : This is the primary Zookeeper configuration file. See the official Zookeeper Configuration Parameters documentation for details on what parameters can be customized. It's not necessary to enter the cluster's node list in this file. That's done automatically by the script during cluster creation.

   - **log4j.properties** : This file sets the default logging levels for Zookeeper components. You can also customize these at the node level when a cluster is created.

   - **zk-supervisord.conf** : The Zookeeper daemon is run under supervision so that if it shuts down unexpectedly, it's automatically restarted by Supervisord. There is nothing much to customize here, but you can refer to the Supervisord Configuration documentation if you want to learn more about the options.

3. Open the image configuration file (in this example, `./zk-image1/zk-image1.conf`) in a text editor. Enter or edit values of configuration properties as required. Properties that must be entered or changed from their default values are marked as **REQUIRED**:

- `DISTRIBUTION_FOR_IMAGE`

  Specify either Ubuntu 14.04 or Debian 8 to use for this image. This guide has *not* been tested on any other versions or distributions.

  All nodes of all clusters created from this image will have this distribution. The default value is `124` corresponding to Ubuntu 14.04 LTS 64-bit. For Debian 8 64-bit, change this value to `140`.

  > **Note**
  >
  > The values represented in this guide are current as of publication, but are subject to change in the future. You can run **`~/storm-linode/linode_api.py distributions`** to see a list of all available distributions and their values in the API.

- `LABEL_FOR_IMAGE`

  A label to help you differentiate this image from others. This name will be shown if you edit or view your images in the Linode Manager.

- `KERNEL_FOR_IMAGE`

  The kernel version provided by Linode to use in this image. The default value is 138, corresponding to the latest 64-bit kernel provided by Linode. It is recommended that you leave this as the default setting.

- `DATACENTER_FOR_IMAGE`

  The Linode data center where this image will be created. This can be any Linode data center, but cluster creation is faster if the image is created in the same data center where the cluster will be created. It's also recommended to create the image in the same data center as the Cluster Manager Linode. Select a data center that is geographically close to your premises, to reduce network latency. If left unchanged, the Linode will be created in the Newark data center.

  This value can either be the data center's ID or location or abbreviation. To see a list of all data centers:

```
./zookeeper-cluster-linode.sh datacenters api_env_linode.conf
```

- `IMAGE_ROOT_PASSWORD` - **REQUIRED**

  The default root user password for the image. All nodes of any clusters created from this image will have this as the root password, unless it's overridden in a cluster's configuration file.

- `IMAGE_ROOT_SSH_PUBLIC_KEY` and `IMAGE_ROOT_SSH_PRIVATE_KEY`

  The keypair files for SSH public key authentication as root user. Any user who logs in with this private key can be authenticated as `root`.

By default, the `cluster_manager.sh` setup has already created a keypair named `clusterroot` and `clusterroot.pub` under `~/.ssh/`. If you wish to replace these with your own keypair, you may create your own keys and set their full paths here.

- `IMAGE_DISABLE_SSH_PASSWORD_AUTHENTICATION`

  This disables SSH password authentication and allows only key based SSH authentication for the cluster nodes. Password authentication is considered less secure, and is hence disabled by default. To enable password authentication, you can change this value to `no`.

- `IMAGE_ADMIN_USER`

  Administrators or developers may have to log in to the cluster nodes for maintenance. Instead of logging in as root users, it's better to log in as a privileged non-root user. The script creates a privileged user with this name in the image (and in all cluster nodes based on this image).

- `IMAGE_ADMIN_PASSWORD` - **REQUIRED**

  Sets the password for the `IMAGE_ADMIN_USER`.

- `IMAGE_ADMIN_SSH_AUTHORIZED_KEYS`

  A file that contains public keys of all personnel authorized to log in to cluster nodes as `IMAGE_ADMIN_USER`. This file should be in the same format as the standard SSH *authorized_keys* file. All the entries in this file are appended to the image's `authorized_keys` file, and get inherited into all nodes based on this image.

  By default, the `cluster_manager.sh` setup creates a new `clusteradmin` keypair, and this variable is set to the path of the public key. You can either retain this generated keypair and distribute the generated private key file `~/.ssh/clusteradmin` to authorized personnel. Alternatively, you can collect public keys of authorized personnel and append them to `~/.ssh/clusteradmin.pub`.

- `IMAGE_DISK_SIZE`

  The size of the image disk in MB. The default value of 5000MB is generally sufficient, since the installation only consists of the OS with Java and Zookeeper software installed.

- `UPGRADE_OS`

  If `yes`, the distribution's packages are updated and upgraded before installing any software. It is recommended to leave the default setting to avoid any installation or dependency issues.

- `INSTALL_ZOOKEEPER_DISTRIBUTION`

  The Zookeeper version to install. By default, `cluster_manager.sh` has already downloaded version 3.4.6. If you wish to install a different version, download it manually and change this variable. However, it is recommended to leave the default value as this guide has not been tested against other versions.

- `ZOOKEEPER_INSTALL_DIRECTORY`

  The directory where Zookeeper will be installed on the image (and on all cluster nodes created from this image).

- `ZOOKEEPER_USER`

  The username under which the Zookeeper daemon runs. This is a security feature to avoid privilege escalation by exploiting some vulnerability in the Zookeeper daemon.

- `ZOOKEEPER_MAX_HEAP_SIZE`

  The maximum Java heap size for the JVM hosting the Zookeeper daemon. This value can be either a percentage, or a fixed value. If the fixed value is not suffixed with any character, it is interpreted as bytes. If it is suffixed with `K`, `M`, or `G`, it is interpreted as kilobytes, megabytes, or gigabytes, respectively.

  If this is too low, it may result in out of memory errors, and cause data losses or delays in the Storm cluster. If it is set too high, the memory for the OS and its processes will be limited, resulting in disk thrashing, which will have a significant negative impact on Zookeeper's performance.

  The default value is 75%, which means at most 75% of the Linode's RAM can be reserved for the JVM, and remaining 25% for the rest of the OS and other processes. It is strongly recommended not to change this default setting.

- `ZOOKEEPER_MIN_HEAP_SIZE`

  The minimum Java heap size to commit for the JVM hosting the Zookeeper daemon. This value can be either a percentage, or a fixed value. If the fixed value is not suffixed with any character, it is interpreted as bytes. If it is suffixed with `K`, `M`, or `G`, it is interpreted as kilobytes, megabytes, or gigabytes, respectively.

  If this value is lower than `ZOOKEEPER_MAX_HEAP_SIZE`, this amount of memory is *committed*, and additional memory up to `ZOOKEEPER_MAX_HEAP_SIZE` is allocated only when the JVM requests it from OS. This can lead to memory allocation delays during operation. So do not set it too low.

  This value should never be more than `ZOOKEEPER_MAX_HEAP_SIZE`. If it is, the Zookeeper daemon will not start.

The default value is 75%, which means 75% of the Linode's RAM is committed – not just reserved – to the JVM and unavailable to any other process. It is strongly recommended not to change this default setting.

When you've finished making changes, save and close the editor.

4. Create the image using `create-image` command, specifying the name of the newly created image and the API environment file:

```
./zookeeper-cluster-linode.sh create-image zk-image1 api_env_linode.conf
```

If the image is created successfully, the output will look something like this at the end:

```
Deleting the temporary linode xxxxxx

Finished creating Zookeeper template image yyyyyy
```

If the process fails, ensure that you do not already have an existing Linode with the same name in the Linode Manager. If you do, delete it and run the command again, or recreate this image with a different name.

> **Note**
>
> During this process, a temporary, short-lived 2GB Linode is created and deleted. This will entail a small cost in your monthly invoice and trigger an event notification email to be sent to the address you have registered with Linode. This is expected behavior.

Create a Zookeeper Cluster

In this section, you will learn how to create a new Zookeeper cluster in which every node is a replica of an existing Zookeeper image. If you have not already created a Zookeeper image, do so first by following **Create a Zookeeper image**.

1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

```
ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
cd storm-linode
```

2. Choose a unique name for your cluster and create a configuration directory using the `new-cluster-conf` command. In this example, we'll call our new cluster configuration `zk-cluster1`:

```
./zookeeper-cluster-linode.sh new-cluster-conf zk-cluster1
```

3. Open the newly created `zk-cluster1.conf` file and make changes as described below. Properties that must be entered or changed from their default values are marked as **REQUIRED**:

- `DATACENTER_FOR_CLUSTER`

  The Linode data center is where the nodes of this cluster will be created. All nodes of a cluster have to be in the same data center; they cannot span multiple data centers since they will use private network traffic to communicate.

  This can be any Linode data center, but cluster creation may be faster if it is created in the same data center where the image and Cluster Manager Linode are created. It is recommended to select a data center that is geographically close to your premises to reduce network latency.
  This value can either be the data center's ID or location or abbreviation. To see a list of all data centers:

  ```
  ./zookeeper-cluster-linode.sh datacenters api_env_linode.conf
  ```

- `CLUSTER_SIZE`

  The types and number of nodes that constitute this cluster. The syntax is:

  `plan:count plan:count ...`

  A `plan` is one of `2GB | 4GB | ... | 120GB` (see Linode plans for all plans) and `count` is the number of nodes with that plan.

  Examples:
  - For a cluster with three 4GB nodes:

    ```
    CLUSTER_SIZE="4GB:3"
    ```

  - For a cluster with three nodes of different plans:

    ```
    CLUSTER_SIZE="2GB:1 4GB:1 8GB:1"
    ```

  The total number of nodes in a Zookeeper cluster **must** be an odd number. Although cluster can have nodes of different plans, it's recommended to use the same plan for all nodes.
  It is recommended to avoid very large clusters. A cluster with 3 to 9 nodes is sufficient for most use cases; 11 to 19 nodes would be considered "large". Anything more than 19 nodes would be counterproductive, because at that point Zookeeper would slow down all the Storm clusters that depend on it.

  Size the cluster carefully, because as of version 3.4.6, Zookeeper does not support dynamic expansion. The only way to resize would be to take it down and create a new cluster, creating downtime for any Storm clusters that depend on it.

- `ZK_IMAGE_CONF` - **REQUIRED**

  Path of the Zookeeper image directory or configuration file to use as a template for creating nodes of this cluster. Every node's disk will be a replica of this image.

The path can either be an absolute path, or a path that is relative to the cluster configuration directory. Using our example, the absolute path would be `/home/clustermgr/storm-linode/zk-image1` and the relative path would be `../zk-image1`.

- `NODE_DISK_SIZE`

  Size of each node's disk in MB. This must be at least as large as the selected image's disk, otherwise the image will not copy properly.

- `NODE_ROOT_PASSWORD`

  Optionally, you can specify a root password for the nodes. If this is empty, the root password will be the `IMAGE_ROOT_PASSWORD` in the image configuration file.

- `NODE_ROOT_SSH_PUBLIC_KEY and NODE_ROOT_SSH_PRIVATE_KEY`

  Optionally, you can specify a custom SSH public key file and private key file for root user authentication. If this is empty, the keys will be the keys specified in image configuration file.

  If you wish to specify your own keypair, select a descriptive filename for this new keypair (example: zkcluster1root), generate them using `ssh-keygen`, and set their full paths here.

- `PUBLIC_HOST_NAME_PREFIX`

  Every Linode in the cluster has a public IP address, which can be reached from anywhere on the Internet, and a *private IP address*, which can be reached only from other nodes of the same user inside the same data center.

  Accordingly, every node is given a public hostname that resolves to its public IP address. Each node's public hostname will use this value followed by a number (for example, `public-host1`, `public-host2`, etc.) If the cluster manager node is in a different Linode data center from the cluster nodes, it uses the public hostnames and public IP addresses to communicate with cluster nodes.

- `PRIVATE_HOST_NAME_PREFIX`

  Every Linode in the cluster is given a *private hostname* that resolves to its private IP address. Each node's private hostname will use this value followed by a number (for example, private-host1, private-host2, etc.). All the nodes of a cluster communicate with one another through their private hostnames. This is also the actual hostname set for the node using the host's `hostname` command and saved in `/etc/hostname`.

- `CLUSTER_MANAGER_USES_PUBLIC_IP`

  Set this value to `false` if the cluster manager node is located in the same Linode data center as the cluster nodes. This is the recommended value. Change to `true` **only** if the cluster manager node is located in a *different* Linode data center from the cluster nodes.

> **Important**
>
> It's important to set this correctly to avoid critical cluster creation failures.

- `ZOOKEEPER_LEADER_CONNECTION_PORT`

  The port used by a Zookeeper node to connect its followers to the leader. When a new leader is elected, each follower opens a TCP connection to the leader at this port. There's no need to change this unless you plan to customize the firewall.

- `ZOOKEEPER_LEADER_ELECTION_PORT`

  The port used for Zookeeper leader election during quorum. There's no need to change this, unless you plan to customize the firewall.

- `IPTABLES_V4_RULES_TEMPLATE`

  Absolute or relative path of the IPv4 iptables firewall rules file. Modify this if you plan to customize the firewall configuration.

- `IPTABLES_V6_RULES_TEMPLATE`

  Absolute or relative path of the IPv6 iptables firewall rules file. IPv6 is completely disabled on all nodes, and no services listen on IPv6 addresses. Modify this if you plan to customize the firewall configuration.

When you've finished making changes, save and close the editor.

4. Create the cluster using the `create` command:

```
./zookeeper-cluster-linode.sh create zk-cluster1 api_env_linode.conf
```

If the cluster is created successfully, a success message is printed:

```
Zookeeper cluster successfully created
```

Details of the created cluster can be viewed using the `describe` command:

```
./zookeeper-cluster-linode.sh describe zk-cluster1
```

Cluster nodes are shut down soon after creation. They are started only when any of the Storm clusters starts.

Create a Storm Image

A *Storm image* is a master disk with all necessary Storm software and libraries downloaded and installed. The benefits of creating a Storm image include:

- Quick creation of a Storm cluster by simply cloning it to create as many nodes as required, each a perfect copy of the image
- Distribution packages and third party software packages are identical on all nodes, and prevent version mismatch errors
- Reduced network usage, because downloads and updates are executed only once when preparing the image, instead of repeating them on each node

> **Note**
>
> If a Storm image already exists, this step is not mandatory. Multiple Storm clusters can share the same Zookeeper image. In fact, it's a good idea to keep the number of images low because image storage is limited to 10GB.
>
> When creating an image, you should have **`clustermgr`** authorization to the Cluster Manager Linode.

1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

2. Choose a unique name for your image and create a configuration directory for the new image using `new-image-conf` command. In this example, we'll call our new image `storm-image1` :

   ```
   ./storm-cluster-linode.sh new-image-conf storm-image1
   ```

   This creates a directory named storm-image1 containing the files that make up the image configuration:

   - **storm-image1.conf** : This is the main image configuration file, and the one you'll be modifying the most. Its properties are described in later steps.

   The other files are secondary configuration files. They contain reasonable default values, but you can always open them in an editor and modify them to suit your needs:

   - **template-storm.yaml** : The Storm configuration file. See the official Storm Configuration documentation for details on what parameters can be customized.

   - **template-storm-supervisord.conf** : The Storm daemon is run under supervision so that if it shuts down unexpectedly, it's automatically restarted by Supervisord. There is nothing much to customize here, but review the Supervisord Configuration documentation if you do want to customize it.

3. Open the image configuration file (in this example, `~/storm-linode/storm-image1/storm-image1.conf`) in a text editor. Enter or edit the values of configuration properties as required. Properties that must be entered or changed from their default values are marked as REQUIRED:

    - `DISTRIBUTION_FOR_IMAGE`

      Specify either Ubuntu 14.04 or Debian 8 to use for this image. This guide has *not* been tested on any other versions or distributions.

      All nodes of all clusters created from this image will have this distribution. The default value is `124` corresponding to Ubuntu 14.04 LTS 64-bit. For Debian 8 64-bit, change this value to `140`.

      > **Note**
      >
      > The values represented in this guide are current as of publication, but are subject to change in the future. You can run **`~/storm-linode/linode_api.py distributions`** to see a list of all available distributions and their values in the API.

    - `LABEL_FOR_IMAGE`

      A label to help you differentiate this image from others. This name will be shown if you edit or view your images in the Linode Manager.

    - `KERNEL_FOR_IMAGE`

      The kernel version provided by Linode to use in this image. The default value is `138` corresponding to the latest 64-bit kernel provided by Linode. It is recommended that you leave this as the default setting.

    - `DATACENTER_FOR_IMAGE`

      The Linode data center where this image will be created. This can be any Linode data center, but cluster creation is faster if the image is created in the same data center where the cluster will be created. It's also recommended to create the image in the same data center as the Cluster Manager Linode. Select a data center that is geographically close to you to reduce network latency.

      This value can either be the data center's ID or location or abbreviation. To see a list of all data centers:

      ```
      ./zookeeper-cluster-linode.sh datacenters api_env_linode.conf
      ```

    - `IMAGE_ROOT_PASSWORD` - **REQUIRED**

      The default root user password for the image. All nodes of any clusters created from this image will have this as the root password, unless it's overridden in a cluster's configuration file.

- `IMAGE_ROOT_SSH_PUBLIC_KEY` and `IMAGE_ROOT_SSH_PRIVATE_KEY`

  The keypair files for SSH public key authentication as root user. Any user who logs in with this private key can be authenticated as root.

  By default, the `cluster_manager.sh` setup has already created a keypair named `clusterroot` and `clusterroot.pub under ~/.ssh/`. If you wish to replace them with your own keypair, you may create your own keys and set their full paths here.

- `IMAGE_DISABLE_SSH_PASSWORD_AUTHENTICATION`

  This disables SSH password authentication and allows only key based SSH authentication for the cluster nodes. Password authentication is considered less secure, and is hence disabled by default. To enable password authentication, you can change this value to `no`.

- `IMAGE_ADMIN_USER`

  Administrators or developers may have to log in to the cluster nodes for maintenance. Instead of logging in as root users, it's better to log in as a privileged non-root user. The script creates a privileged user with this name in the image (and in all cluster nodes based on this image).

- `IMAGE_ADMIN_PASSWORD` - **REQUIRED**

  Sets the password for the `IMAGE_ADMIN_USER`.

- `IMAGE_ADMIN_SSH_AUTHORIZED_KEYS`

  A file that contains public keys of all personnel authorized to log in to cluster nodes as `IMAGE_ADMIN_USER`. This file should be in the same format as the standard SSH *authorized_keys* file. All the entries in this file are appended to the image's `authorized_keys` file, and get inherited into all nodes based on this image.

  By default, the `cluster_manager.sh` setup creates a new clusteradmin keypair, and this variable is set to the path of the public key. You can either retain this generated keypair and distribute the generated private key file `~/.ssh/clusteradmin` to authorized personnel. Alternatively, you can collect public keys of authorized personnel and append them to `~/.ssh/clusteradmin.pub`.

- `IMAGE_DISK_SIZE`

  The size of the image disk in MB. The default value of 5000MB is generally sufficient, since the installation only consists of the OS with Java and Storm software installed.

- `UPGRADE_OS`

  If yes, the distribution's packages are updated and upgraded before installing any software. It is recommended to leave the default setting to avoid any installation or dependency issues.

- `INSTALL_STORM_DISTRIBUTION`

  The Storm version to install. By default, the `cluster_manager.sh` setup has already downloaded version 0.9.5. If you wish to install a different version, download it manually and change this variable. However, it is recommended to leave the default value as this guide has not been tested against other versions.

- `STORM_INSTALL_DIRECTORY`

  The directory where Storm will be installed on the image (and on all cluster nodes created from this image).

- `STORM_YAML_TEMPLATE`

  The path of the template `storm.yaml` configuration file to install in the image. By default, it points to the `template-storm.yaml` file under the image directory. Administrators can either customize this YAML file before creating the image, or set this variable to point to another `storm.yaml` of their choice.

- `STORM_USER`

  The username under which the Storm daemon runs. This is a security feature to avoid privilege escalation by exploiting some vulnerability in the Storm daemon.

- `SUPERVISORD_TEMPLATE_CONF`

  The path of the template supervisor configuration file to install in the image. By default, it points to the `template-storm-supervisord.conf` file in the Storm image directory. Administrators can modify this file before creating the image, or set this variable to point to any other `storm-supervisord.conf` file of their choice.

Once you've made changes, save and close the editor.

4. Create the image using the `create-image` command, specifying the name of the newly created image and the API environment file:

```
./storm-cluster-linode.sh create-image storm-image1 api_env_linode.conf
```

If the image is created successfully, the output will look something like this towards the end:

```
....
Deleting the temporary linode xxxxxx

Finished creating Storm template image yyyyyy
```

If the process fails, ensure that you do not already have an existing Storm image with the same name in the Linode Manager. If you do, delete it and run the command again, or recreate this image with a different name.

> **Note**
>
> During this process, a short-lived 2GB Linode is created and deleted. This will entail a small cost in the monthly invoice and trigger an event notification email to be sent to the address you have registered with Linode. This is expected behavior.

Create a Storm Cluster

In this section, you will learn how to create a new Storm cluster in which every node is a replica of an existing Storm image. If you have not created any Storm images, do so first by following **Create a Storm image**.

> **Note**
>
> When creating a cluster, you should have clustermgr authorization to the Cluster Manager Linode.

1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

2. Choose a unique name for your cluster and create a configuration directory using the `new-cluster-conf` command. In this example, we'll call our new cluster configuration `storm-cluster1`:

   ```
   ./storm-cluster-linode.sh new-cluster-conf storm-cluster1
   ```

   This creates a directory named `storm-cluster1` that contains the main configuration file, `storm-cluster1.conf`, which will be described in the next step. If you chose a different name when you ran the previous command, your directory and configuration file will be named accordingly.

3. Open the newly created `storm-cluster1.conf` file and make changes as described below. Properties that must be entered or changed from their default values are marked as REQUIRED:

   - `DATACENTER_FOR_CLUSTER`

     The Linode data center where the nodes of this cluster will be created. All nodes of a cluster have to be in the same data center; they cannot span multiple data centers since they will use private network traffic to communicate.

     This can be any Linode data center, but cluster creation may be faster if it is created in the same data center where the image and Cluster Manager Linode are created. It is recommended to select a data center that is geographically close to your premises to reduce network latency.

     This value can either be the data center's ID or location or abbreviation. To see a list of all data centers:

     ```
     ./zookeeper-cluster-linode.sh datacenters api_env_linode.conf
     ```

- `NIMBUS_NODE`

  This specifies the Linode plan to use for the Nimbus node, which is responsible for distributing and coordinating a Storm topology to supervisor nodes.

  It should be one of `2GB | 4GB | ... | 120GB` (see Linode plans for all plans). The default size is 2GB, but a larger plan is strongly recommended for the Nimbus node.

- `SUPERVISOR_NODES`

  Supervisor nodes are the workhorses that execute the spouts and bolts that make up a Storm topology.

  The size and number of supervisor nodes should be decided based on how many topologies the cluster should run concurrently, and the computational complexities of their spouts and bolts. The syntax is:

  `plan:count plan:count ...`

  A `plan` is one of `2GB | 4GB| ....| 120GB` (see Linode plans for all plans) and `count` is the number of supervisor nodes with that plan. Although a cluster can have supervisor nodes of different sizes, it's recommended to use the same plan for all nodes.

  The number of supervisor nodes can be increased later using the `add-nodes` command (see Expand Cluster).

  Examples:

  - Create three 4GB nodes:

  ```
  SUPERVISOR_NODES="4GB:3"
  ```

  - Create six nodes with three different plans:

  ```
  SUPERVISOR_NODES="2GB:2 4GB:2 8GB:2"
  ```

- `CLIENT_NODE`

  The client node of a cluster is used to submit topologies to it and monitor it. This should be one of `2GB | 4GB | ... | 120GB` (see Linode plans for all plans). The default value of 2GB is sufficient for most use cases.

- `STORM_IMAGE_CONF` - **REQUIRED**

  Path of the Storm image directory or configuration file to use as a template for creating nodes of this cluster. Every node's disk will be a replica of this image.

The path can either be an absolute path, or a path that is relative to this cluster configuration directory. Using our example, the absolute path would be `/home/clustermgr/storm-linode/storm-image1` and the relative path would be `../storm-image1`.

- `NODE_DISK_SIZE`

  Size of each node's disk in MB. This must be at least as large as the selected image's disk, otherwise the image will not copy properly.

- `NODE_ROOT_PASSWORD`

  Optionally, you can specify a root password for the nodes. If this is empty, the root password will be the `IMAGE_ROOT_PASSWORD` in the image configuration file.

- `NODE_ROOT_SSH_PUBLIC_KEY` and `NODE_ROOT_SSH_PRIVATE_KEY`

  Optionally, you can specify a custom SSH public key file and private key file for root user authentication. If this is empty, the keys will be the keys specified in image configuration file.

  If you wish to specify your own keypair, select a descriptive filename for this new keypair (example: *zkcluster1root*), generate them using `ssh-keygen`, and set their full paths here.

- `NIMBUS_NODE_PUBLIC_HOSTNAME`, `SUPERVISOR_NODES_PUBLIC_HOSTNAME_PREFIX` and `CLIENT_NODES_PUBLIC_HOSTNAME_PREFIX`

  Every Linode in the cluster has a *public IP address*, which can be reached from anywhere on the Internet, and a *private IP address*, which can be reached only from other nodes of the same user inside the same data center.

  Accordingly, every node is given a *public hostname* that resolves to its public IP address. Each node's public hostname will use this value followed by a number (for example, `public-host1`, `public-host2`, etc.) If the cluster manager node is in a different Linode data center from the cluster nodes, it uses the public hostnames and public IP addresses to communicate with cluster nodes.

- `NIMBUS_NODE_PRIVATE_HOSTNAME`, `SUPERVISOR_NODES_PRIVATE_HOSTNAME_PREFIX` and `CLIENT_NODES_PRIVATE_HOSTNAME_PREFIX`

  Every Linode in the cluster is given a *private hostname* that resolves to its private IP address. Each node's private hostname will use this value followed by a number (for example, `private-host1`, `private-host2`, etc.). All the nodes of a cluster communicate with one another through their private hostnames. This is also the actual hostname set for the node using the host's `hostname` command and saved in `/etc/hostname`.

- `CLUSTER_MANAGER_USES_PUBLIC_IP`

  Set this value to `false` if the cluster manager node is located in the *same* Linode data center as the cluster nodes. This is the recommended value and is also the default. Change to `true` only if the cluster manager node is located in a *different* Linode data center from the cluster nodes.

  > **Important**
  >
  > It's important to set this correctly to avoid critical cluster creation failures.

- `ZOOKEEPER_CLUSTER` - **REQUIRED**

  Path of the Zookeeper cluster directory to be used by this Storm cluster.

  This can be either an absolute path or a relative path that is relative to this Storm cluster configuration directory. Using our example, the absolute path would be `/home/clustermgr/storm-linode/zk-cluster1`, and the relative path would be `../zk-cluster1`.

- `IPTABLES_V4_RULES_TEMPLATE`

  Absolute or relative path of the IPv4 iptables firewall rules file applied to Nimbus and Supervisor nodes. Modify this if you plan to customize their firewall configuration.

- `IPTABLES_CLIENT_V4_RULES_TEMPLATE`

  Absolute or relative path of the IPv4 iptables firewall rules file applied to Client node. Since the client node hosts a cluster monitoring web server and should be accessible to administrators and developers, its rules are different from those of other nodes. Modify this if you plan to customize its firewall configuration.

  Default: `../template-storm-client-iptables-rules.v4`

- `IPTABLES_V6_RULES_TEMPLATE`

  Absolute or relative path of the IPv6 iptables firewall rules file followed for all nodes, including client node. IPv6 is completely disabled on all nodes, and no services listen on IPv6 addresses. Modify this if you plan to customize the firewall configuration.

  When you've finished making changes, save and close the editor.

4. Create the cluster using the `create` command:

   ```
   ./storm-cluster-linode.sh create storm-cluster1 api_env_linode.conf
   ```

   If the cluster is created successfully, a success message is printed:

   ```
   Storm cluster successfully created
   ```

   Details of the created cluster can be viewed using the `describe` command:

```
./storm-cluster-linode.sh describe storm-cluster1
```

Cluster nodes are shut down soon after creation.

Start a Storm Cluster

This section will explain how to start a Storm cluster. Doing so will also start any Zookeeper clusters on which it depends, so they do not need to be started separately.

> **Note**
>
> When starting a cluster, you should have **clustermgr** authorization to the Cluster Manager Linode.

1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

2. Start the Storm cluster using the `start` command. This example uses the `storm-cluster1` naming convention from above, but if you chose a different name you should replace it in the command:

   ```
   ./storm-cluster-linode.sh start storm-cluster1 api_env_linode.conf
   ```

3. If cluster is being started for the very first time, see the next section for how to authorize users to monitor a Storm cluster.

Monitor a Storm Cluster

Every Storm cluster's client node runs a Storm UI web application for monitoring that cluster, but it can be accessed only from allowed workstations.

The next two sections explain how to allow workstations and monitor a cluster from the web interface.

Allow Workstations to Monitor a Storm Cluster
When performing the steps in this section, you should have `clustermgr` authorization to the Cluster Manager Linode.

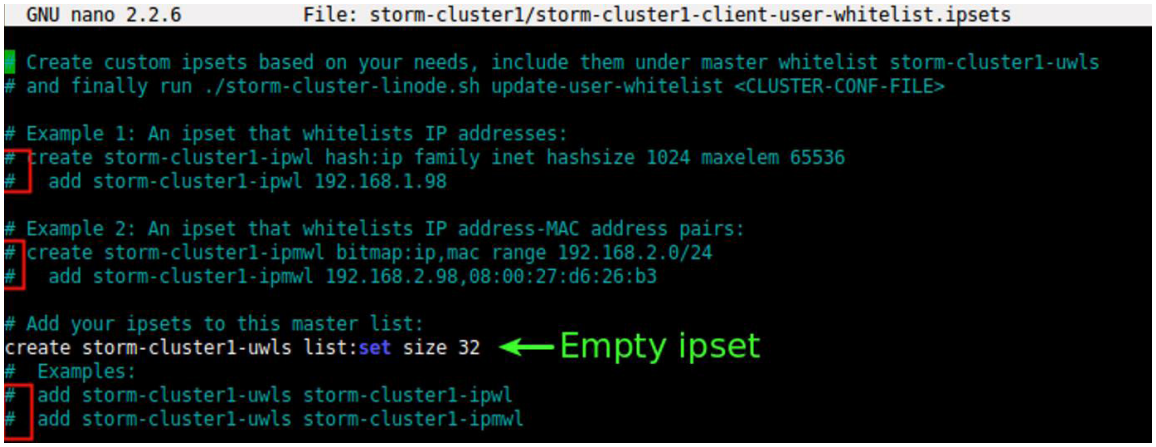
1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

2. Open the `your-cluster/your-cluster-client-user-whitelist.ipsets` file (using our example from above, `storm-cluster1/storm-cluster1-client-user-whitelist.ipsets`) file in a text editor.

   This file is an ipsets list of allowed IP addresses. It consists of one master ipset and multiple child ipsets that list allowed machines by IP addresses or other attributes such as MAC IDs.

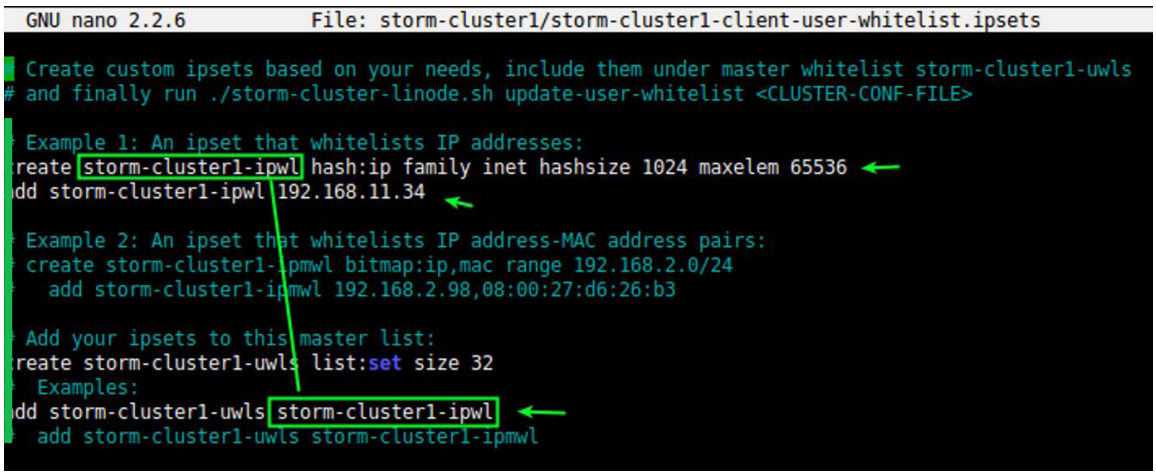
The master ipset is named your-cluster-uwls. By default, it's completely empty, which means nobody is authorized.



3. To allow an IP address:

   - Uncomment the line that creates the *your-cluster-ipwl* ipset
   - Add the IP address under it
   - Add *your-cluster-ipwl* to the master *ipset your-cluster-uwls*

   These additions are highlighted below:



> **Note**
>
> Any IP address that is being included in the file should be a public facing IP address of the network. For example, company networks often assign local addresses like 10.x.x.x or 192.x.x.x addresses to employee workstations, which are then NATted to a public IP address while sending requests outside the company network. Since the cluster client node is in the Linode cloud outside your company network, it will see monitoring requests as arriving from this public IP address. So it's the public IP address that should be allowed.

4. Any number or type of additional ipsets can be created, as long as they are added to the master ipset.

See the **Set Types** section in the ipset manual for available types of ipsets. Note that some of the types listed in the manual may not be available on the client node because the ipset version installed on it using Ubuntu or Debian package manager is likely to be an older version.

5. Enter all required ipsets, save the file, and close the editor.

6. Activate the new ipsets with the `update-user-whitelist` command:

```
./storm-cluster-linode.sh update-user-whitelist storm-cluster1
```

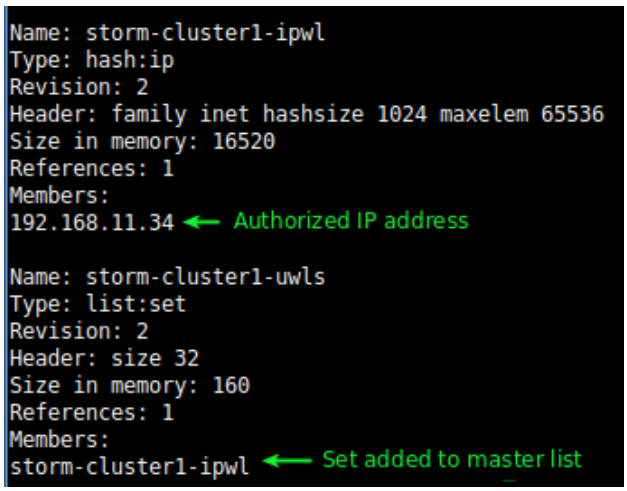
7. Log in to the client node from the Cluster Manager Linode:

```
ssh -i ~/.ssh/clusterroot root@storm-cluster1-private-client1
```

Verify that the new ipsets have been configured correctly:

```
ipset list
```

You should see output similar to the following (in addition to custom ipsets if you added them, and the ipsets for the Storm and Zookeeper cluster nodes):



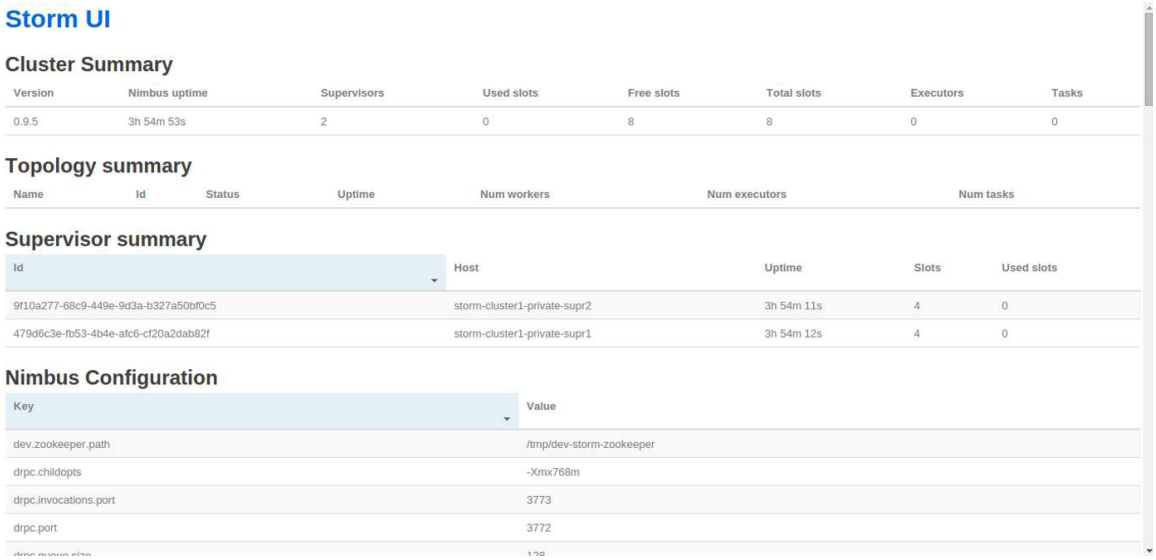
Disconnect from the client node and navigate back to the `storm-linode` directory on the cluster manager node:

```
exit
```

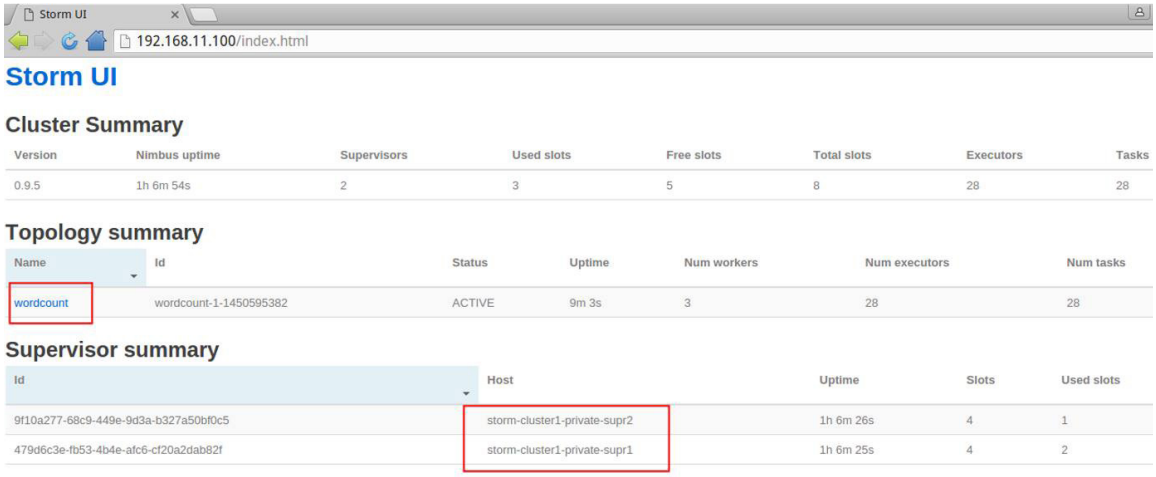
8. From the cluster manager node, get the public IP address of the client node. This IP address should be provided to users authorized to access the Storm UI monitoring web application. To show the IP addresses, use the `describe` command:

```
./storm-cluster-linode.sh describe storm-cluster1
```

9. Finally, verify that the Storm UI web application is accessible by opening `http://public-IP-of-client-node` in a web browser on each whitelisted workstation. You should see the Storm UI web application, which looks like this:



The Storm UI displays the list of topologies and the list of supervisors executing them:



If the cluster is executing any topologies, they are listed under the **Topology summary** section. Click on a topology to access its statistics, supervisor node logs, or actions such as killing that topology.

Test a New Storm Cluster

1. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

```
ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
cd storm-linode
```

2. Get the private IP address of the client node of the target cluster. This is preferred for security and to minimize impact on the data transfer quota, but the public IP address works as well:

```
./storm-cluster-linode.sh describe storm-cluster1
```

3. Log in to the client node as its `IMAGE_ADMIN_USER` user (the default is `clusteradmin`, configured in the Storm image configuration file) via SSH using an authorized private key:

```
ssh -i ~/.ssh/clusteradmin clusteradmin@192.168.42.13
```

4. Run the following commands to start the preinstalled word count example topology:

```
cd /opt/apache-storm-0.9.5/bin
./storm jar ../examples/storm-starter/storm-starter-topologies-0.9.5.jar storm.starter.
WordCountTopology “wordcount”
```

5. A successful submission should produce output similar to this:

```
Running: java -client -Dstorm.options= -Dstorm.home=/opt/apache-storm-0.9.5 -Dstorm.
log.dir=/var/log/storm -Djava.library.path=/usr/local/lib:/opt/local/lib:/usr/lib -Dstorm.
conf.file= -cp /opt/apache-storm-0.9.5/lib/disruptor-2.10.1.jar:/opt/apache-storm-0.9.5/
lib/minlog-1.2.jar:/opt/apache-storm-0.9.5/lib/commons-io-2.4.jar:/opt/apache-storm-0.9.5/
lib/clj-time-0.4.1.jar:/opt/apache-storm-0.9.5/lib/clout-1.0.1.jar:/opt/apache-storm-0.9.5/
lib/ring-devel-0.3.11.jar:/opt/apache-storm-0.9.5/lib/tools.macro-0.1.0.jar:/opt/apache-
storm-0.9.5/lib/ring-jetty-adapter-0.3.11.jar:/opt/apache-storm-0.9.5/lib/jetty-util-6.1.26.
jar:/opt/apache-storm-0.9.5/lib/commons-exec-1.1.jar:/opt/apache-storm-0.9.5/lib/tools.
cli-0.2.4.jar:/opt/apache-storm-0.9.5/lib/objenesis-1.2.jar:/opt/apache-storm-0.9.5/
lib/jetty-6.1.26.jar:/opt/apache-storm-0.9.5/lib/ring-servlet-0.3.11.jar:/opt/apache-
storm-0.9.5/lib/storm-core-0.9.5.jar:/opt/apache-storm-0.9.5/lib/hiccup-0.3.6.jar:/opt/
apache-storm-0.9.5/lib/clojure-1.5.1.jar:/opt/apache-storm-0.9.5/lib/commons-codec-1.6.
jar:/opt/apache-storm-0.9.5/lib/servlet-api-2.5.jar:/opt/apache-storm-0.9.5/lib/compo-
jure-1.1.3.jar:/opt/apache-storm-0.9.5/lib/json-simple-1.1.jar:/opt/apache-storm-0.9.5/lib/
commons-logging-1.1.3.jar:/opt/apache-storm-0.9.5/lib/math.numeric-tower-0.0.1.jar:/opt/
apache-storm-0.9.5/lib/asm-4.0.jar:/opt/apache-storm-0.9.5/lib/commons-lang-2.5.jar:/opt/
apache-storm-0.9.5/lib/clj-stacktrace-0.2.2.jar:/opt/apache-storm-0.9.5/lib/kryo-2.21.
jar:/opt/apache-storm-0.9.5/lib/logback-classic-1.0.13.jar:/opt/apache-storm-0.9.5/lib/
slf4j-api-1.7.5.jar:/opt/apache-storm-0.9.5/lib/reflectasm-1.07-shaded.jar:/opt/apache-
storm-0.9.5/lib/ring-core-1.1.5.jar:/opt/apache-storm-0.9.5/lib/joda-time-2.0.jar:/opt/
apache-storm-0.9.5/lib/logback-core-1.0.13.jar:/opt/apache-storm-0.9.5/lib/snakeyaml-1.11.
jar:/opt/apache-storm-0.9.5/lib/carbonite-1.4.0.jar:/opt/apache-storm-0.9.5/lib/tools.log-
ging-0.2.3.jar:/opt/apache-storm-0.9.5/lib/core.incubator-0.1.0.jar:/opt/apache-storm-0.9.5/
lib/chill-java-0.3.5.jar:/opt/apache-storm-0.9.5/lib/jgrapht-core-0.9.0.jar:/opt/apache-
storm-0.9.5/lib/jline-2.11.jar:/opt/apache-storm-0.9.5/lib/commons-fileupload-1.2.1.
jar:/opt/apache-storm-0.9.5/lib/log4j-over-slf4j-1.6.6.jar:../examples/storm-starter/
storm-starter-topologies-0.9.5.jar:/opt/apache-storm-0.9.5/conf:/opt/apache-storm-0.9.5/
bin -Dstorm.jar=../examples/storm-starter/storm-starter-topologies-0.9.5.jar storm.start-
er.WordCountTopology wordcount

1038 [main] INFO  backtype.storm.StormSubmitter - Jar not uploaded to master yet. Sub-
mitting jar...
1061 [main] INFO  backtype.storm.StormSubmitter - Uploading topology jar ../examples/
storm-starter/storm-starter-topologies-0.9.5.jar to assigned location: /var/lib/storm/
```

```
nimbus/inbox/stormjar-3a9e3c47-88c3-44c2-9084-046f31e57668.jar
Start uploading file '../examples/storm-starter/storm-starter-topologies-0.9.5.jar' to '/
var/lib/storm/nimbus/inbox/stormjar-3a9e3c47-88c3-44c2-9084-046f31e57668.jar' (3248678
bytes)
[==================================================] 3248678 / 3248678
File '../examples/storm-starter/storm-starter-topologies-0.9.5.jar' uploaded to '/var/lib/
storm/nimbus/inbox/stormjar-3a9e3c47-88c3-44c2-9084-046f31e57668.jar' (3248678 bytes)
1260 [main] INFO  backtype.storm.StormSubmitter - Successfully uploaded topology jar
to assigned location: /var/lib/storm/nimbus/inbox/stormjar-3a9e3c47-88c3-44c2-9084-
046f31e57668.jar
1261 [main] INFO  backtype.storm.StormSubmitter - Submitting topology wordcount in
distributed mode with conf {"topology.workers":3,"topology.debug":true}
2076 [main] INFO  backtype.storm.StormSubmitter - Finished submitting topology: word-
count
```

6. Verify that the topology is running correctly by opening the Storm UI in a web browser. The "wordcount" topology should be visible in the **Topology Summary** section.

The above instructions will use the sample "wordcount" topology, which doesn't provide a visible output to show the results of the operations it is running. However, this topology simply counts words in generated sentences, so the number under "Emitted" is the actual word count.

For a more practical test, feel free to download another topology, such as the Reddit Comment Sentiment Analysis Topology, which outputs a basic list of threads within given subreddits, based upon which have the most positive and negative comments over time. If you do choose to download a third-party topology, be sure it is from a trustworthy source and that you download it to the correct directory.

Start a New Topology

If you or a developer have created a topology, perform these steps to start a new topology on one of your Linode Storm clusters:

> **Note**
>
> The developer should have **`clusteradmin`** (or **`clusterroot`**) authorization to log in to the client node of the target Storm cluster.
>
> Optionally, to get the IP address of client node, the developer should have **`clustermgrguest`** (or **`clustermgrroot`**) authorization to log in to the Cluster Manager Linode. If the IP address is known by other methods, this authorization is not required.

1. Package your topology along with all the third-party classes on which they depend into a single JAR (Java Archive) file.

2. If multiple clusters are deployed, select the target Storm cluster to run the topology on. Get the public IP address of the client node of the target cluster. See **cluster description** for details on how to do this.

3. Transfer the topology JAR from your local workstation to client node:

```
scp -i ~/.ssh/private-key local-topology-path clusteradmin@public-ip-of-client-node:
topology-jar
```

Substitute `private-key` for the private key of the Storm client, `local-topology-path` for the local filepath of the JAR file, `PUBLIC-IP-OF-CLIENT-NODE` for the IP address of the Storm client, and `topology-jar` for the filepath you'd like to use to store the topology on the client node.

4. Log in to the client node as `clusteradmin`, substituting the appropriate values:

```
ssh -i ~/.ssh/private-key clusteradmin@PUBLIC-IP-OF-CLIENT-NODE
```

5. Submit the topology to the cluster:

```
cd /opt/apache-storm-0.9.5/bin
./storm jar topology-jar.jar main-class arguments-for-topology
```

Substitute `topology-jar.jar` for the path of the JAR file you wish to submit, main-class with the main class of the topology, and `arguments-for-topology` for the arguments accepted by the topology's main class.

6. **Monitor the execution of the new topology.**

> **Note**
>
> The Storm UI will show only information on the topology's execution, not the actual data it is processing. The data, including its output destination, is handled in the topology's JAR files.

Other Storm Cluster Operations

In this section, we'll cover additional operations to manage your Storm cluster once it's up and running.

All commands in this section should be performed from the `storm-linode` directory on the cluster manager Linode. You will need `clustermgr` privileges unless otherwise specified.

Expand a Storm Cluster
If the supervisor nodes of a Storm cluster are overloaded with too many topologies or other CPU-intensive jobs, it may help to add more supervisor nodes to alleviate some of the load.

Expand the cluster using the `add-nodes` command, specifying the plans and counts for the new nodes. For example, to add three new 4GB supervisor nodes to a cluster named `storm-cluster1`:

```
./storm-cluster-linode.sh add-nodes storm-cluster1 api_env_linode.conf "4GB:3"
```

Or, to add a 2GB and two 4GB supervisor nodes to `storm-cluster1`:

```
./storm-cluster-linode.sh add-nodes storm-cluster1 api_env_linode.conf "2GB:1 4GB:2"
```

This syntax can be used to add an arbitrary number of different nodes to an existing cluster.

Describe a Storm Cluster
A user with `clustermgr` authorization can use `describe` command to describe a Storm cluster:

```
./storm-cluster-linode.sh describe storm-cluster1
```

A user with only `clustermgrguest` authorization can use `cluster_info.sh` to describe a Storm cluster using `list` to get a list of names of all clusters, and the `info` command to describe a given cluster. When using the `info` command, you must also specify the cluster's name:

```
./cluster_info.sh list
./cluster_info.sh info storm-cluster1
```

Stop a Storm Cluster
Stopping a Storm cluster stops all topologies executing on that cluster, stops Storm daemons on all nodes, and shuts down all nodes. The cluster can be restarted later. Note that the nodes will still incur hourly charges even when stopped.

To stop a Storm cluster, use the `stop` command:

```
./storm-cluster-linode.sh stop storm-cluster1 api_env_linode.conf
```

Destroy a Storm Cluster
Destroying a Storm cluster permanently deletes all nodes of that cluster and *their* data. They will no longer incur hourly charges.

To destroy a Storm cluster, use the `destroy` command:

```
./storm-cluster-linode.sh destroy storm-cluster1 api_env_linode.conf
```

Run a Command on all Nodes of a Storm Cluster
You can run a command (for example, to install a package or download a resource) on all nodes of a Storm cluster. This is also useful when updating and upgrading software or changing file permissions. Be aware that when using this method, the command will be executed as `root` on each node.

To execute a command on all nodes, use the `run` command, specifying the cluster name and the commands to be run. For example, to update your package repositories on all nodes in `storm-cluster1` :

```
./storm-cluster-linode.sh run storm-cluster1 “apt-get update”
```

Copy Files to all Nodes of a Storm Cluster
You can copy one or more files from the cluster manager node to all nodes of a Storm cluster. The files will be copied as the `root` user on each node, so keep this in mind when copying files that need specific permissions.

1. If the files are not already on your cluster manager node, you will first need to copy them from your workstation. Substitute `local-file` for the name or path of the file on your local machine, and `PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE` for the IP address of the cluster manager node. You can also specify a different filepath and substitute it for `~`:

   ```
   scp -i ~/.ssh/clustermgr local-files clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE:~
   ```

2. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

3. Execute the `cp` command, specifying the destination directory on each node and the list of local files to copy:

   ```
   ./storm-cluster-linode.sh cp target-cluster-name “target-directory” “local-files”
   ```

   Remember to specify the target directory before the list of source files (this is the reverse of regular `cp` or `scp` commands).

   For example, if your topology requires data files named “*.data” for processing, you can copy them to `root` user’s home directory on all cluster nodes with:

   ```
   ./storm-cluster-linode.sh cp storm-cluster1 “~” “~/*.data”
   ```

Delete a Storm Image
To delete a Storm image, use the `delete-image` command:

```
./storm-cluster-linode.sh delete-image storm-image1 api_env_linode.conf
```

Note that this command will delete the image, but not any clusters that were created from it.

Zookeeper Cluster Operations

In this section, we’ll cover additional operations to manage your Zookeeper cluster once it’s up and running.

All commands in this section should be performed from the `storm-linode` directory on the cluster manager Linode. You will need `clustermgr` privileges unless otherwise specified.

Describe a Zookeeper Cluster
A user with `clustermgr` authorization can use the describe command to describe a Zookeeper cluster:

```
./zookeepercluster-linode.sh describe zk-cluster1
```

A user with only `clustermgrguest` authorization can use `cluster_info.sh` to describe a Zookeeper cluster using `list` to get a list of names of all clusters, and the `info` command to describe a given cluster. When using the `info` command, you must specify the cluster’s name:

```
./cluster_info.sh list
./cluster_info.sh info zk-cluster1
```

Stop a Zookeeper Cluster
Stopping a Zookeeper cluster cleanly stops the Zookeeper daemon on all nodes, and shuts down all nodes. The cluster can be restarted later. Note that the nodes **will** still incur Linode's hourly charges when stopped.

> **Important**
>
> Do not stop a Zookeeper cluster while any Storm clusters that depend on it are running. This may result in data loss.

To stop a cluster, use the `stop` command:

```
./zookeeper-cluster-linode.sh stop zk-cluster1 api_env_linode.conf
```

Destroy a Zookeeper Cluster
Destroying a Zookeeper cluster permanently deletes all nodes of that cluster and their data. Unlike a Linode that is only shut down, destroyed or deleted Linodes no longer incur hourly charges.

> **Important**
>
> Do not destroy a Zookeeper cluster while any Storm clusters that depend on it are running. It may result in data loss.

To destroy a cluster, use the `destroy` command:

```
./zookeeper-cluster-linode.sh destroy zk-cluster1 api_env_linode.conf
```

Run a Command on all Nodes of a Zookeeper Cluster

You can run a command on all nodes of a Zookeeper cluster at once. This can be useful when updating and upgrading software, downloading resources, or changing permissions on new files. Be aware that when using this method, the command will be executed as root on each node.

To execute a command on all nodes, use the run command, specifying the cluster name and the commands to be run. For example, to update your package repositories on all nodes:

```
./zookeeper-cluster-linode.sh run zk-cluster1 “apt-get update”
```

Copy Files to all Nodes of a Zookeeper Cluster

You can copy one or more files from the cluster manager node to all nodes of a Storm cluster. The files will be copied as the `root` user on each node, so keep this in mind when copying files that need specific permissions.

1. If the files are not already on your cluster manager node, you will first need to copy them from your workstation. Substitute `local-file` for the name or path of the file on your local machine, and `cluster-manager-IP` for the IP address of the cluster manager node. You can also specify a different filepath and substitute it for `~`:

   ```
   scp -i ~/.ssh/clustermgr local-files clustermgr@cluster-manager-IP:~
   ```

2. Log in to the Cluster Manager Linode as `clustermgr` and navigate to the `storm-linode` directory:

   ```
   ssh -i ~/.ssh/clustermgr clustermgr@PUBLIC-IP-OF-CLUSTER-MANAGER-LINODE
   cd storm-linode
   ```

3. Execute the `cp` command, specifying the destination directory on each node and the list of local files to copy:

   ```
   ./zookeeper-cluster-linode.sh cp target-cluster-name “target-directory” “local-files”
   ```

   Remember to specify the target directory before the list of source files (this is the reverse of regular `cp` or `scp` commands).

   For example, if your cluster requires data files named “*.data” for processing, you can copy them to `root` user’s home directory on all cluster nodes with:

   ```
   ./zookeeper-cluster-linode.sh cp zk-cluster1 “~” “~/*.data”
   ```

Delete a Zookeeper Image

To delete a Zookeeper image, execute the `delete-image` command:

```
./zookeeper-cluster-linode.sh delete-image zk-image1 api_env_linode.conf
```

Note that this command will delete the image, but not any clusters that were created from it.

More Information

You may wish to consult the following resources for additional information on this topic. While these are provided in the hope that they will be useful, please note that we cannot vouch for the accuracy or timeliness of externally hosted materials.

- Apache Storm project website
- Apache Storm documentation
- Storm - Distributed and Fault-Tolerant Real-time Computation

About Akamai Cloud Computing

Akamai accelerates innovation with scalable, simple, affordable, and accessible Linux cloud solutions and services. Our products, services, and people give developers and enterprises the flexibility, support, and trust they need to build, deploy, secure, and scale applications more easily and cost-effectively from cloud to edge on the world's most distributed network.

www.akamai.com
www.linode.com

Cloud Computing Services Developers Trust

linode.com | Support: 855-4-LINODE | Sales: 844-869-6072
249 Arch St., Philadelphia, PA 19106 Philadelphia, PA 19106