

The Practical Guide to GitOps

Deliver quality at speed



Table of Contents

The Freedom of Choice	03	GitOps Hands On Tutorial	11
The Principles of GitOps	04	Part 1: Spin up a Kubernetes Cluster	14
Key Benefits of GitOps	05	Part 2: Fork the Sock Shop Repository	21
What Happens When you Adopt GitOps?	06	Part 3: Setup CI and Connect a Container Registry	23
Typical CI/CD Pipeline	07	Part 4: Let's Get Started with GitOps	31
GitOps Separation of Privileges	10	Further Resources	36

GitOps in Practice

A “you build it, you own it” development process requires tools that developers know and understand. GitOps is our name for how we use developer tooling to drive operations.

GitOps is a way to do Continuous Delivery. More specifically, it is an operating model for building Cloud Native applications that unifies Deployment, Monitoring and Management. It works by using Git as a source of truth for declarative infrastructure and applications. Automated CI/CD pipelines roll out changes to your infrastructure when commits are pushed and approved in Git. It also makes use of diff tools to compare the actual production state with what’s under source control and alerts you when there is a divergence.

The ultimate goal of GitOps is to speed up development so that your team can make changes and updates safely and securely to complex applications running in Kubernetes.

Freedom of choice

Because there is no single tool that can do everything required in your pipeline, GitOps gives you the freedom to choose the best tools for the different parts of your CI/CD pipeline. You can select a set of tools from the open source ecosystem or from closed source or depending on your use case, you may even combine them. The most difficult part of creating a pipeline is gluing all of the pieces together.

Whatever you choose for your delivery pipeline, applying GitOps best practices with Git (or any version control) should be an integral component of your process. It will make building and adopting continuous delivery in your organization easier and will significantly speed up your team’s ability to ship features.

The Principles of GitOps

1 The entire system is described declaratively.

Kubernetes is just one example of many modern cloud native tools that are “declarative” and that can be treated as code. Declarative means that configuration is guaranteed by a set of facts instead of by a set of instructions. With your application’s declarations versioned in Git, you have a single source of truth. Your apps can then be easily deployed and rolled back to and from Kubernetes. And even more importantly, when disaster strikes, your cluster’s infrastructure can also be dependably and quickly reproduced.

2 The canonical desired system state is versioned in Git.

With the declaration of your system stored in a version control system, and serving as your canonical source of truth, you have a single place from which everything is derived and driven. This trivializes rollbacks; where you can use a ``git revert`` to go back to your previous application state. With Git’s excellent security guarantees, you can also use your SSH key to sign commits that enforce strong security guarantees about the authorship and provenance of your code.

3 Approved changes to the desired state are automatically applied to the system.

Once you have the declared state kept in Git, the next step is to allow any changes to that state to be automatically applied to your system. What’s significant about this is that you don’t need cluster credentials to make a change to your system. With GitOps, there is a segregated environment that the state definition lives outside of. This allows you to separate what you do and how you’re going to do it.

4 Software agents ensure correctness and alert on divergence

Once the state of your system is declared and kept under version control, software agents can inform you whenever reality doesn’t match your expectations. The use of agents also ensures that your entire system is self-healing. And by self-healing, we don’t just mean when nodes or pods fail—those are handled by Kubernetes—but in a broader sense, like in the case of human error. In this case, software agents act as the feedback and control loop for your operations.

Key Benefits of GitOps

1. Increased Productivity

Continuous deployment automation with an integrated feedback control loop speeds up your mean time to deployment. This allows your team to ship 30-100x more changes per day, and increases overall development output 2-3 times.

2. Enhanced Developer Experience

Push code and not containers. By applying GitOps best practices developers use familiar tools like Git to manage updates and features to Kubernetes more rapidly. Newly on boarded developers can get quickly up to speed and be productive within days instead of months.

3. Improved Compliance and Stability

When you use Git to manage your Kubernetes cluster, you automatically gain a convenient audit log of all cluster changes outside of Kubernetes. An audit trail of who did what, and when to your cluster can be used to meet SOC 2 compliance and ensure stability.

4. Higher Reliability

With Git's capability to revert/rollback and fork, you gain stable and reproducible rollbacks. Because your entire system is described in Git, you have a single source of truth from which to recover after a meltdown, reducing your meantime to recovery (MTTR) from hours to minutes.

5. Increased Consistency and Standardization

Because GitOps provides one model for making infrastructure, apps and add on changes, you have consistent end to end workflows across your entire organization. Not only are your continuous integration and continuous deployment pipelines all driven by pull request, but your operations tasks are also fully reproducible through Git.

6. Stronger Security Guarantees

Git's strong correctness and security guarantees, backed by the strong cryptography used to track and manage changes, as well as the ability to sign changes to prove authorship and origin is key to a secure definition of the desired state of the cluster.

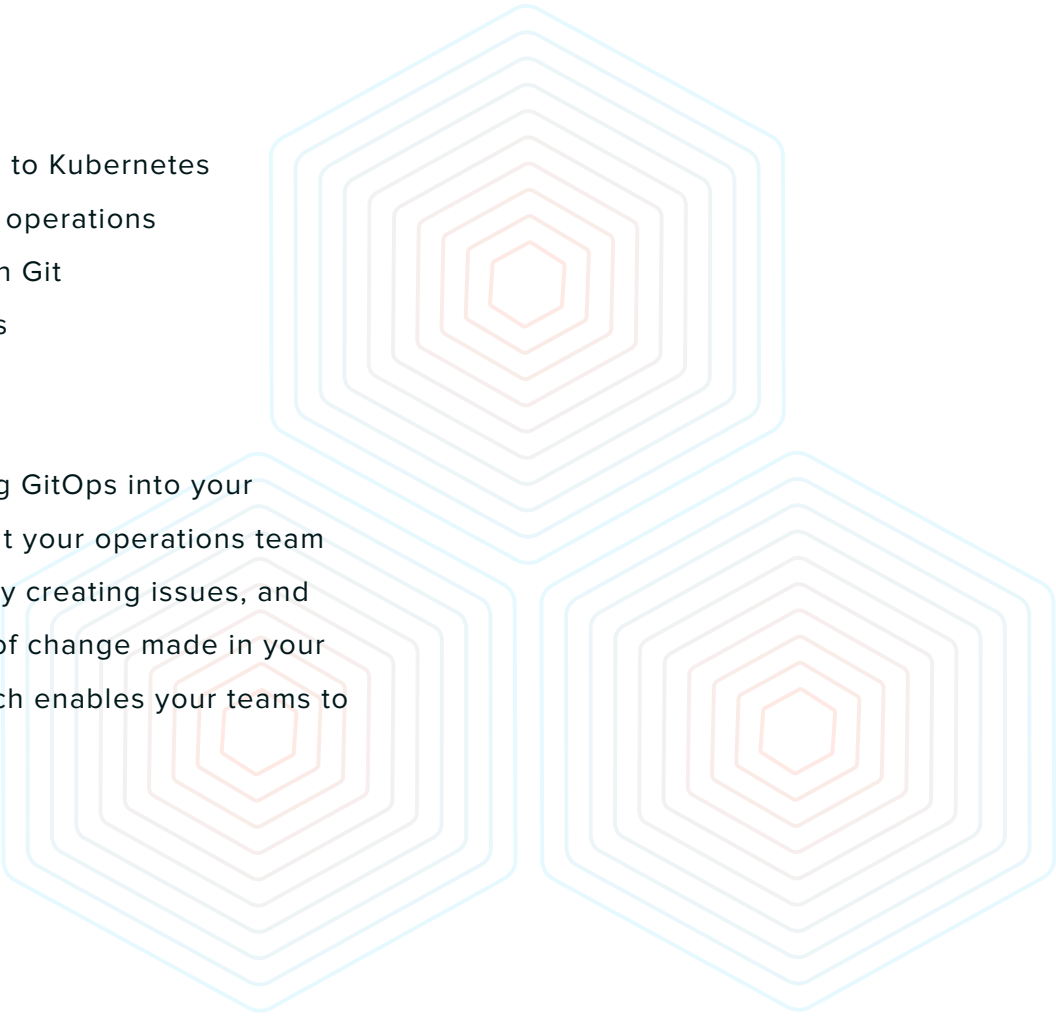
What Happens When you Adopt GitOps?



Introducing GitOps into your organization means:

- ➔ Any developer that uses Git can start deploying new features to Kubernetes
- ➔ The same workflows are maintained across development and operations
- ➔ All changes can be triggered, stored, validated and audited in Git
- ➔ Ops changes can be made by pull request including rollbacks
- ➔ Ops changes can be observed and monitored

Since all of your developers are already living in Git, incorporating GitOps into your organization is simple. Having everything in one place means that your operations team can also use the same workflow to make infrastructure changes by creating issues, and reviewing pull requests. GitOps allows you to roll back any kind of change made in your cluster. In addition to this, you also get built-in observability, which enables your teams to have more autonomy to make changes.

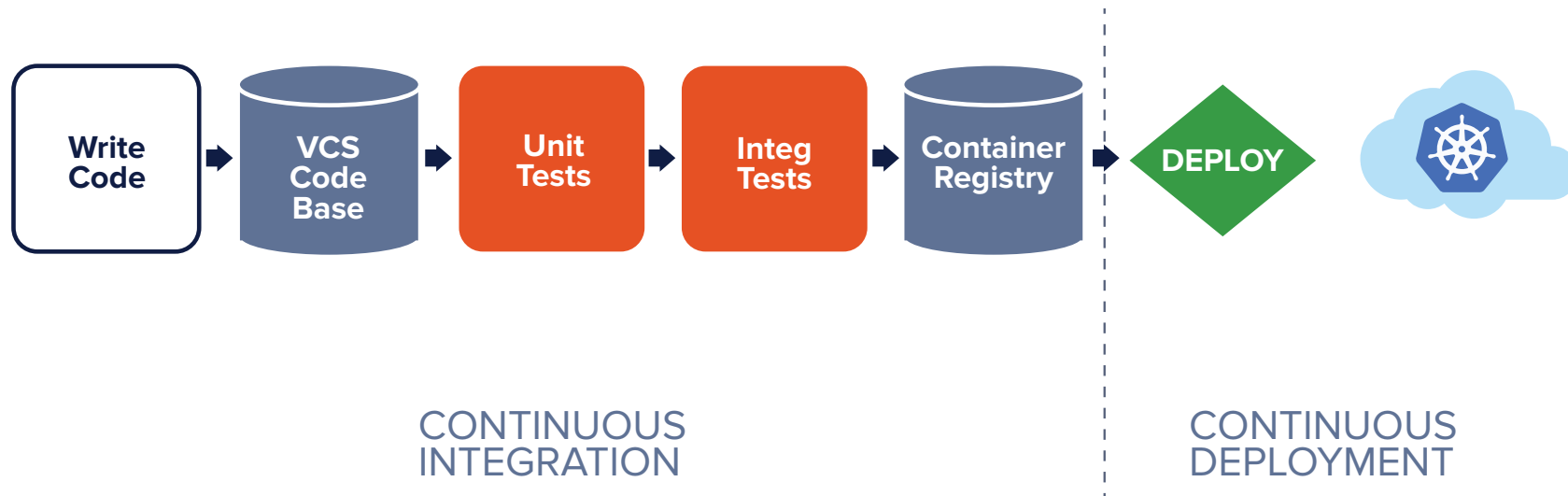


Typical CI/CD Pipeline

What does a typical CI/CD pipeline look like in most organizations? Your development team writes and then pushes code into a code repo. Let's say for example that you have one microservice repo and in that repo, you bundle your application code along with deployment YAML manifest files or a Helm chart that define how your application will run in your cluster. When you push that code to Git, the continuous integration tool kicks off unit tests that eventually build the Docker image that gets pushed to the container registry.

With a typical CI/CD pipeline, Docker images are deployed using some sort of bash script or another method of talking directly to the cluster API.

A TYPICAL SOFTWARE DELIVERY PIPELINE



Security and the Typical CI/CD Pipeline

How secure is the typical CI/CD pipeline?

With this approach, your CI tooling pushes and deploys images to the cluster. For the CI system to apply the changes to a cluster, you have to share your API credentials with the CI tooling and that means your CI tool becomes a high value target. If someone breaks into your CI tool, they will have total control over your production cluster, even if your production cluster is highly secure.

But what happens if your cluster goes down, and you need to recreate it? How do you restore the previous state of your application? You would have to run all of your CI jobs to rebuild everything and then re-apply all of the workloads to the new cluster. The typical CI pipeline doesn't have its state easily recorded.

Let's see how we can improve the typical CI/CD pipeline with GitOps.

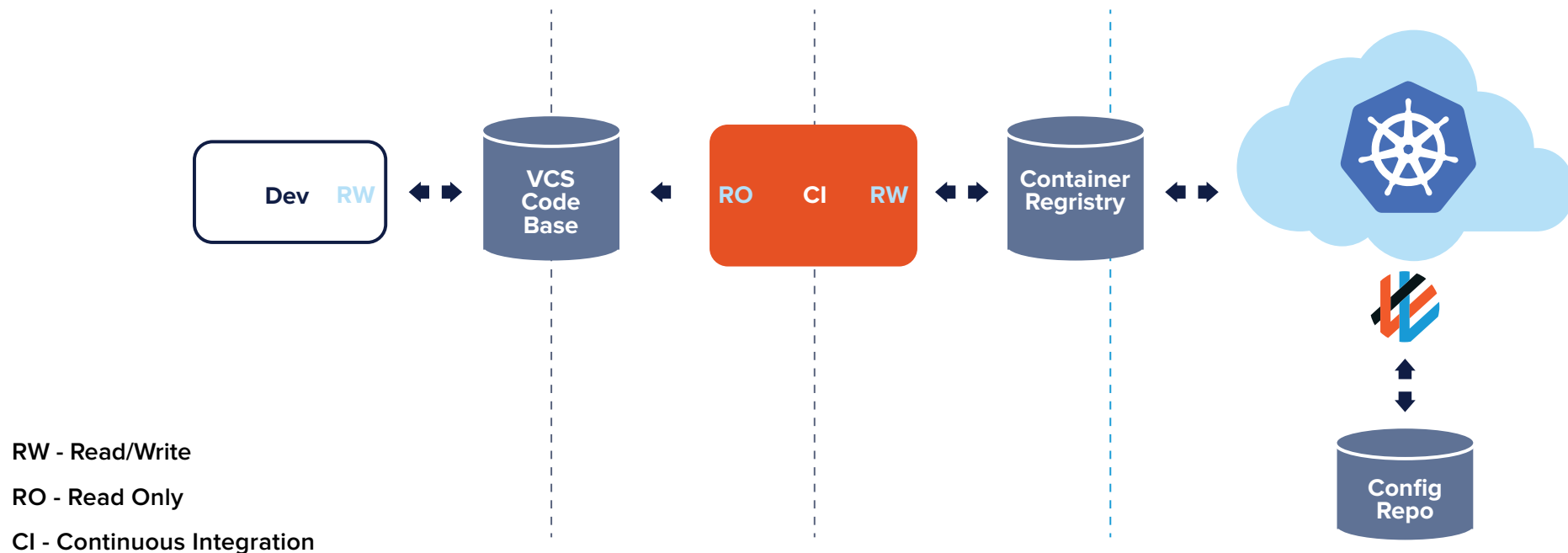


The GitOps Deployment Pipeline

GitOps implements a Kubernetes controller that listens for and synchronizes deployments to your Kubernetes cluster. The controller uses the operator pattern. This is significant on two levels:

1. It is more secure.
2. It automates complex error prone tasks like having to manually update YAML manifests.

With the operator pattern, an agent acts on behalf of the cluster. It listens to events relating to custom resource changes, and then applies those changes based on a deployment policy. The agent is responsible for synchronizing **what's in Git with what's running in the cluster**, providing a simple way for your team to achieve continuous deployment.



GitOps Separation of Privileges

GitOps separates CI from CD and is a more secure method of deploying applications to Kubernetes. The table below shows how GitOps separates read/write privileges between the cluster, your CI and CD tooling and the container repository, providing your team with a secure method of creating updates.

CI Tooling Test, Build, Scan, Publish	CD Tooling Synchronize Git with the cluster
Runs outside the production cluster	Runs inside the production cluster
Read access to the code repository	Read/Write access to configuration repository
Read/Write access to container repository	Read access to image repository
Read/Write access to the continuous integration environment	Read/Write access to the production cluster



GITOPS

HANDS ON TUTORIAL

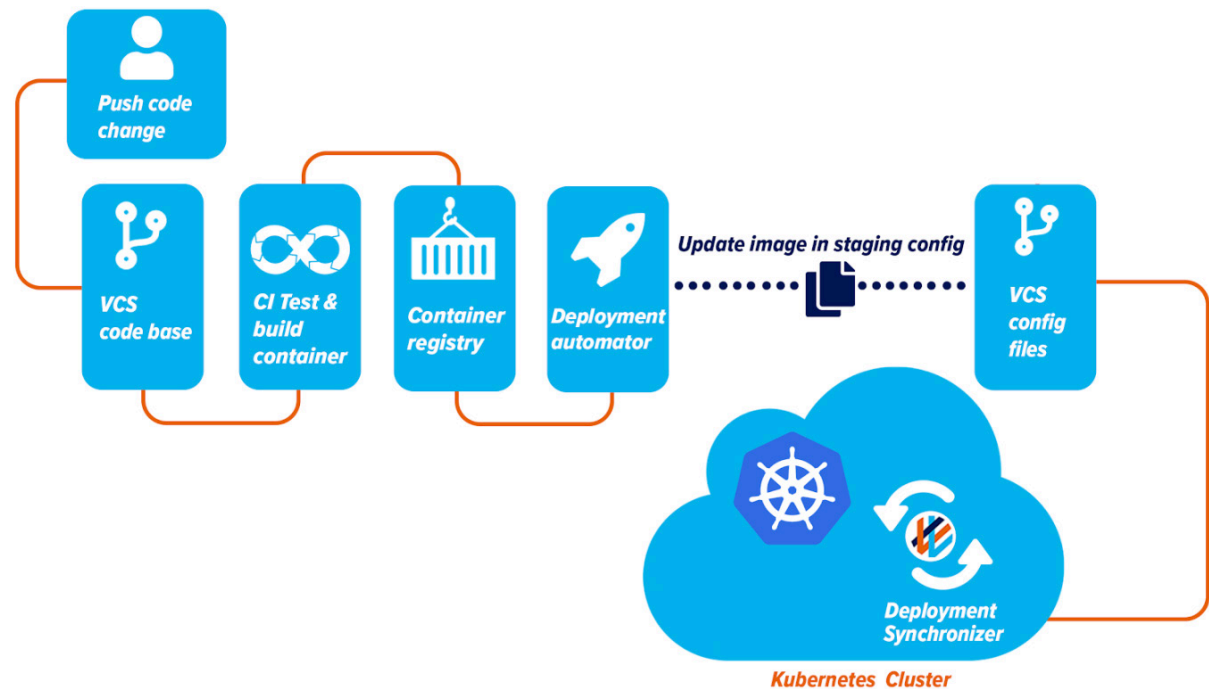
GitOps Hands On Tutorial

In this tutorial, we'll show you how to set up a CI/CD pipeline. We'll then deploy a demo application to a cluster, make a small update to the application and deploy the change with Weave Cloud.

In our product Weave Cloud, the GitOps core machinery is in its CI/CD tooling with the critical piece being continuous deployment (CD) that supports Git-cluster synchronization. This provides a feedback loop that gives you control over your Kubernetes cluster.

Here is a typical developer workflow for creating or updating a new feature:

1. A pull request for a new feature is pushed to Git for review.
2. The code is reviewed and approved by a colleague. Once reviewed and revised, it is merged to Git.
3. The Git merge triggers the CI pipeline that runs a series of tests, builds a new image and pushes the new image to a container registry.
4. Weave Cloud's Deployment Automator watches the container registry, notices the image, pulls the new image from the registry and updates its manifest in the config repo.
5. The Weave Cloud Deployment Synchronizer (installed to the cluster), detects that the cluster is out of date. It pulls the changed manifests from the config repo and deploys the new feature to production.



Tutorial Prerequisites

Before you can start doing GitOps, you'll need to set up the following items. Once complete, you will have an entire end to end CI/CD pipeline setup. In this tutorial we use particular tools, but keep in mind that you can pick and choose the tools you need and you don't have to use the ones listed here.



A Kubernetes cluster

To complete this tutorial, you will need a Kubernetes cluster. This tutorial should work with any valid Kubernetes installation. This example uses three Ubuntu hosts on [Digital Ocean](#) and then installs Kubernetes with [kubeadm](#). Sign up for Digital Ocean to receive a three month free trial (you'll have to input your credit card for this).

There are plenty of other options as well, such as [minikube](#), or you can use one of the public cloud providers like [Google Kubernetes Engine](#) or Amazon's [EKS](#).



[Github](#) account

You'll need to create a new repository for this tutorial and make it available from your cluster through a URL from Git.



[Quay.io](#) account

This is the container repository that we'll use in this tutorial.



[Travis](#) account

This tutorial uses Travis as the continuous integration piece of the pipeline.



[Weave Cloud](#) Trial Account

You'll use Weave Cloud to automate, observe and control your deployments..



PART 1:

SPIN UP A KUBERNETES CLUSTER

Part 1: Spin up a Kubernetes Cluster

1. Create three droplets in Digital Ocean

Sign up or log into [Digital Ocean](#) and create three Ubuntu instances with the following specifications:

- Ubuntu 16.04
- 4GB or more of RAM per instance

2. Set up Kubernetes with kubeadm

Kubeadm is by far the simplest way to set up a Kubernetes cluster. With only a few commands, you can deploy a complete Kubernetes cluster with a resilient and secure container network onto the Cloud Provider of your choice in only a few minutes.

kubeadm is a command line tool that you can use to easily spin up a cluster. In this tutorial, we'll run the minimum commands to get a cluster up and running. For information on all kubeadm command-line options, see the [kubeadm docs](#).

Part 1: Spin up a Kubernetes Cluster

3. Download and install kubelet, kubeadm and Docker

To begin SSH into each machine and become root (for example, run `sudo su -`). Install the required binaries onto all three instances:

```
curl -s https://packages.cloud.google.com/apt/doc/apt-key.gpg | apt-key add -  
cat <<EOF > /etc/apt/sources.list.d/kubernetes.list  
deb http://apt.kubernetes.io/ kubernetes-xenial main  
EOF  
apt-get update
```

Next, download and install Docker:

```
apt-get install -y docker.io
```

And finally, install the Kubernetes packages:

```
apt-get install -y kubelet kubeadm kubectl kubernetes-cni
```


Part 1: Spin up a Kubernetes Cluster

4. Initialize the Master node

Before making one of your machines a master, `kubelet` and `kubeadm` must have been installed onto each of the nodes. Initialize the master, by picking one of the machines and running:

```
kubeadm init
```

The network interface is auto-detected and then advertises the master on it with the default gateway.

A successful initialization outputs the following at the end:

```
kubeadm join --token <token-id> <master-ip>
```

Make a record of the `kubeadm join` command that `kubeadm init` outputs. You will need this once it's time to join the nodes. This token is used for mutual authentication between the master and any joining nodes.

Part 1: Spin up a Kubernetes Cluster

5. Set up the environment for Kubernetes.

On the master run the following as a regular user:

```
sudo cp /etc/kubernetes/admin.conf $HOME/  
sudo chown $(id -u):$(id -g) $HOME/admin.conf  
export KUBECONFIG=$HOME/admin.conf
```

6. Install Weave Net as the Pod Networking Layer

In this section, you will install a Weave Net pod network so that your pods can communicate with each other.

Install Weave Net by running on the master:

```
kubectl apply -f "https://cloud.weave.works/k8s/net?k8s-version=$(kubectl version |  
base64 | tr -d '\n')"
```

Confirm that the pod network is working and that the kube-dns pod is running:

```
kubectl get pods --all-namespaces
```

Once the kube-dns pod is up and running, you can join all of the nodes to form the cluster.

Part 1: Spin up a Kubernetes Cluster

7. Join the Nodes to the Master

The nodes are where the workloads (containers and pods, etc) run.

Join the nodes to your cluster with (shown on the master node that you initialized):

```
kubeadm join --token <token> <master-ip>
```

After the node has successfully joined, you will see the following in the terminal:

Run `kubectl get nodes` on the master to see this machine join.

Run `kubectl get nodes` on the master to display a cluster with the number of machines as you created.

Part 1: Spin up a Kubernetes Cluster

8. Install and Launch the Weave Cloud Agents

Sign up for [Weave Cloud](#) and create a new instance. Weave Cloud instances are the primary workspaces for your application and provides a view onto your cluster and the application that is running on it.

1. Step through the setup screens and choose `Generic Kubernetes`.
2. Run the custom command that appears on your master node. After running the command, you will see several objects created in your cluster.
3. Weave Cloud is now connected to your instances. View your cluster by clicking on the Explore tab.

For automated deployments and GitOps, there is one additional configuration step that we will cover in part two of this tutorial.



PART 2:

FORK THE DEMO APPLICATION REPOSITORY

Part 2: Fork the Demo Application Repository

1. Fork and clone the repositories: microservices-demo and front-end

You will need a [GitHub account](#) for this step.

Before you can modify the microservices demo application, The Sock Shop, fork the following two repositories:

- <https://github.com/microservices-demo/front-end> - This is the front-end service of the Sock Shop application. You will update the color the buttons in this example.
- <https://github.com/microservices-demo/microservices-demo> - This is the repo that stores the Kubernetes configuration files for the application. The Weave Cloud deployment agent automatically updates the front-end YAML manifest file in this repository.



PART 3:

SETUP CI AND CONNECT A CONTAINER REGISTRY

Part 3: Setup CI and Connect a Container Registry

1. Get a Container Registry Account at Quay

Sign up for a [Quay.io account](#), and record the user name. After you log in, you can see it under “Users and Organizations” on the top right of the repositories page.

Create a new public repository called front-end. This is the container repository that will be used by Travis CI to push the new images.

2. Sign up for a [Travis Continuous Integration](#) Account

If you already have your own CI system, you can use that instead. All that Weave Cloud needs is something that creates a container image and pushes it to the registry whenever you push a change to GitHub.

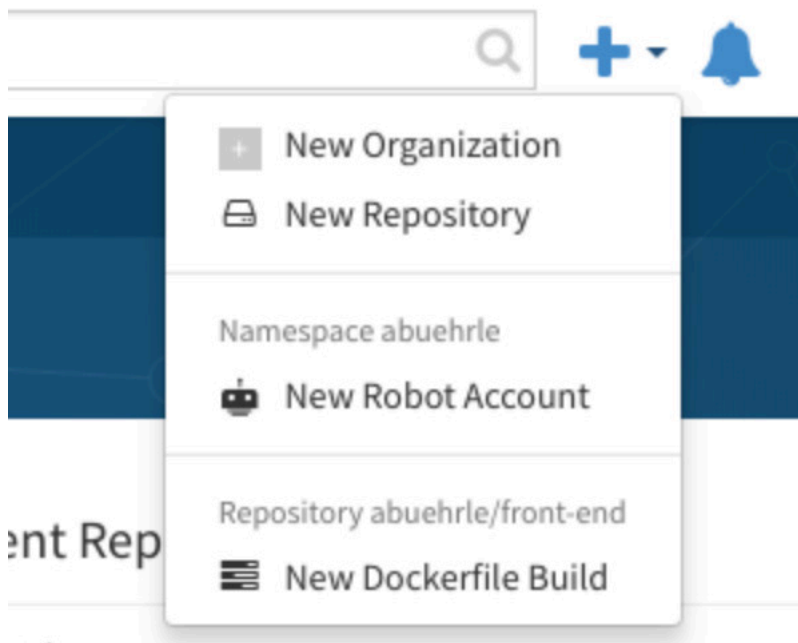
In this example, you will hook up Travis CI to your GitHub and Quay.io accounts.

Connect your GitHub account by clicking the + button next to My Repositories and then toggle the build button for `<YOUR_GITHUB_USERNAME>/front-end` so that Travis can automatically run builds for that repo.

Part 3: Setup CI and Connect a Container Registry

3. Configure a Robot Account in Quay.io and then link it to Travis CI

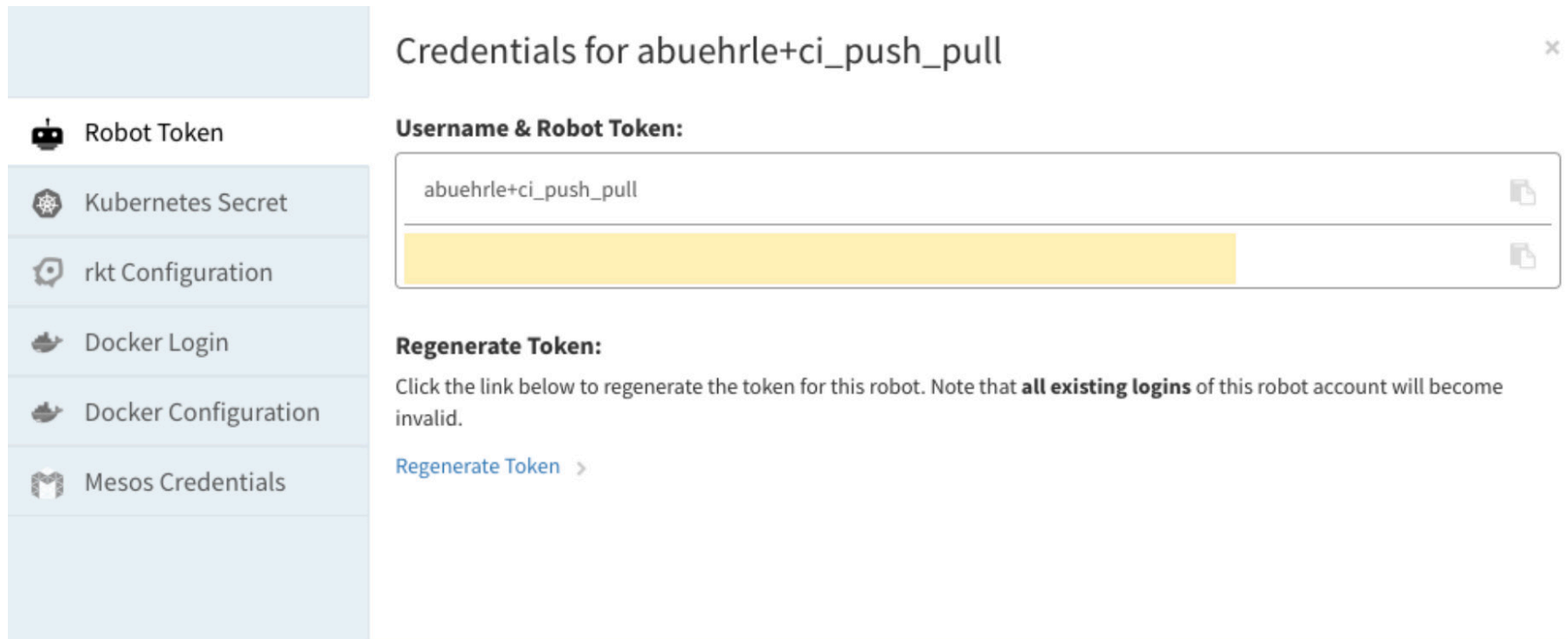
1. Go back to Quay.io, select your repository. Create a Robot Account by selecting the + from the header and call it **ci_push_pull**.



Part 3: Setup CI and Connect a Container Registry

3. Configure a Robot Account in Quay.io and link it to Travis CI (continued)

1. Ensure that the Robot Account has Admin permissions in Quay on your repository.
2. Copy the username and the generated token for the Quay robot account:



The screenshot shows the Quay.io interface for managing robot accounts. On the left is a sidebar with navigation options: Robot Token, Kubernetes Secret, rkt Configuration, Docker Login, Docker Configuration, and Mesos Credentials. The main content area is titled 'Credentials for abuehrle+ci_push_pull'. It contains two sections: 'Username & Robot Token:' which shows the username 'abuehrle+ci_push_pull' and a yellowed-out token, and 'Regenerate Token:' which includes a warning that existing logins will become invalid and a 'Regenerate Token' link.

Part 3: Setup CI and Connect a Container Registry

4. Go back to TravisCI, and select the front-end repo.

Select More **Options** → **Settings**. Toggle 'Build pushed branches' and 'Build pushed pull requests'.



Current Branches Build History Pull Requests Settings

General

☒ Build pushed branches ?

☐ Limit concurrent jobs ?

☒ Build pushed pull requests ?

Part 3: Setup CI and Connect a Container Registry

5. In the same dialog, add your Quay.io Robot Account credentials to the front-end repo in Travis:

```
DOCKER_USER=<"user-name+robot-account">  
DOCKER_PASS=<"robot-token">
```

Where,

<"user-name+ci_push_pull"> is your user-name including the + sign and the name of the robot account.

<"robot-token"> is the token found and copied from the Robot Token dialog box.

Part 3: Setup CI and Connect a Container Registry

6. Edit the travis.yml file

In the env section of the .travis.yml file located in the root of the forked front-end repo add

quay.io/<YOUR_QUAY_USERNAME> (replace with your Quay.io username) to the env: group section of the file:

env:

```
- GROUP=quay.io/<YOUR_QUAY_USERNAME> COMMIT="${TRAVIS_COMMIT}" TAG="${TRAVIS_TAG}"  
REPO=front-end;
```

Commit and push this change to your fork of the front-end repo.

```
git commit -m "Update .travis.yml to refer to my quay.io account." .travis.yml  
git push
```

Part 3: Setup CI and Connect a Container Registry

7. Modify the Manifest file so it Points to Your Container Image

Using an editor of your choice, open `manifests/front-end-dep.yaml`, from the `microservices-demo` repo you forked and update the image line.

Change it from:

```
image: weaveworksdemos/front-end
```

To:

```
image: quay.io/$YOUR_QUAY_USERNAME/front-end:<deploy-tag>
```

Where,

- `$YOUR_QUAY_USERNAME` - your Quay.io username
- `<deploy-tag>` - set this to `master` (although you may want to specify a branch in practice)

Commit and push the change to your GitHub fork:

```
git commit -m "Update front-end to refer to my fork." manifests/front-end-dep.yaml
git push
```

Go back to Travis-CI and watch as the image, as it's unit-tested, built and pushed to Quay.io.




PART 4:

LET'S GET STARTED WITH GITOPS

Part 4: Let's Get Started with GitOps

1. Finish configuring Weave Cloud by connecting your forked repo.



1. Click  then select Deploy from the menu that appears and then follow the instructions that appear.

2. **Config repository** - Add the ssh URL with the path to the YAML files for the microservices-demo:

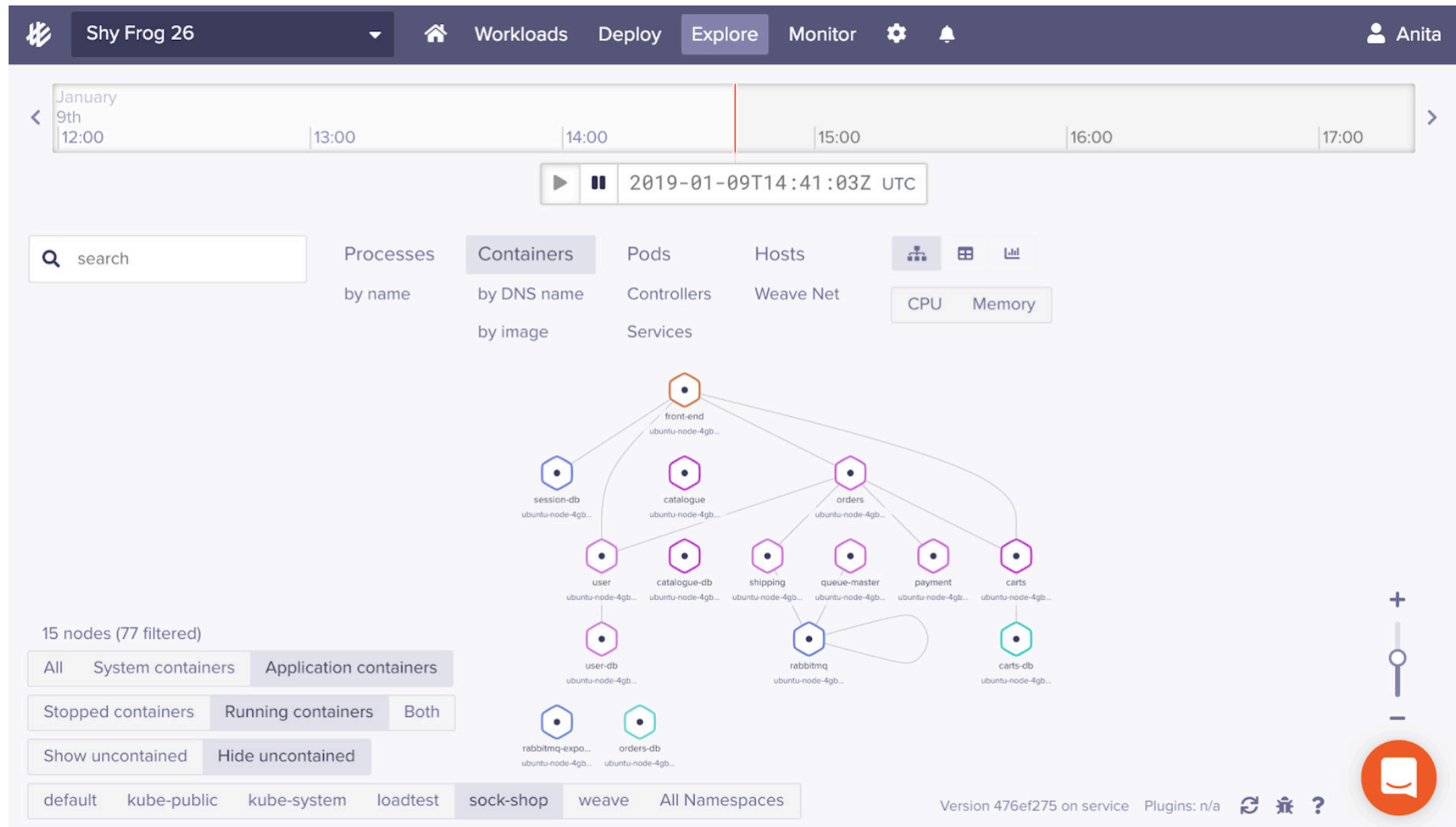
```
SSH URL: git@github.com:<YOUR_GITHUB_USERNAME>/microservices-demo  
Path to YAML files: deploy/kubernetes/manifests  
branch: master
```

3. **Configure the repository read/write permissions** - Push keys to GitHub.

4. Run command that appears on the master node of your cluster.

Go to the Workload screen and watch as all of the services in the Sock Shop get automatically deployed. Go to Explore in Weave Cloud to watch as it deploys to the cluster.

Part 4: Let's Get Started with GitOps



Part 4: Let's Get Started with GitOps

2. View the Sock Shop in Your Browser

Find the port that the cluster allocated for the front-end service by running:

```
kubectl describe svc front-end -n sock-shop
```

Launch the Sock Shop in your browser by going to the IP address of any of your node machines in your browser, and by specifying the NodePort. So for example, `http://<master_ip>:<pNodePort>`. You can find the IP address of the machines in the DigitalOcean dashboard. The nodeport is typically 30001.

Part 4: Let's Get Started with GitOps

3. Make a Change to the Socks Shop and Deploy it

Note the active states of the Catalogue and the Cart buttons are blue. In the next section you will change those to red.

On your workstation, or wherever you have front-end checked out, open up the file `./public/css/style.blue.css` in a text editor and **search and replace** `#4993e4` **with** `red`.

Push the change to Github:

```
git commit -am "Change buttons to red."
git push
```

Watch as the new image deploys and shows up as a new image in the Weave Cloud UI.

Click the Deploy button, and reload the Socks Shop in your browser. Notice that the buttons in the catalogue and on the cart have all changed to red!

So that's useful for manually gated changes, but it's even better to do continuous delivery.

Enable the Continuous Delivery policy by toggling the **'Automate'** button.

Further Resources

You can find much more about GitOps online



[A comprehensive
GitOps primer](#)



[The GitOps FAQ](#)



[A recorded presentation on GitOps
and its implementation
at Weaveworks](#)

About Weaveworks

Weaveworks makes it fast and simple for developers to build and operate containerized applications. The [Weave Cloud](#) operations-as-a-service platform provides a continuous delivery pipeline for building and operating applications, letting teams connect, monitor and manage microservices and containers on any server or public cloud. Weaveworks also contributes to several open source projects, including Weave Scope, Weave Flux and eksctl. It was one of the first members of the Cloud Native Computing Foundation. Founded in 2014, the company is backed by Google Ventures, Accel Partners and others.

Production Ready Kubernetes with Weaveworks

We can help you accelerate your Kubernetes journey with our subscription service that supports installing production-grade Kubernetes on-premise, in AWS and GCP from development to production. For the past 3 years, Kubernetes has been powering Weave Cloud, our operations as a service offering, so we're taking our knowledge and helping teams embrace the benefits of cloud native tooling. We've focused on creating GitOps workflows. Our approach uses developer-centric tooling (e.g. Git) and a tested approach to help you install, setup, operate and upgrade Kubernetes.

[Contact us](#) for more details or learn more about our [professional services](#).

