Who am I?

• Developer Advocate in the AWS container service team
• Previous roles at Red Hat, Mesosphere, MapR, applied research
• Find me via …
  • Slack: AWS dev, Kubernetes, CNCF, Weaveworks
  • Twitter: @mhausenblas
  • Mail: hausenbl@amazon.com
Books

Note: book signing Thu 7 Nov from 10:30 to 11:30
Who are you?

admin
developer
architect
SRE
QA
PM

velocityconf.com
#VelocityConf
Agenda

- Kubernetes and Git 101
- GitOps motivation & model
- **BREAK 10:30am to 11am**
- GitOps in action
- Progressive delivery
- Challenges
Learning goals

• Being able to explain what GitOps is
• Know about the benefit of GitOps
• Having some hands-on experience with an example GitOps set up
• Being able to decide if GitOps is the right choice for team/project
Kubernetes 101
Architecture

Overview
Architecture

API Server

kubectl get deploys
Kubernetes resources

• Core and custom resources
  • spec: kubectl/HTTP API (via params and/or YAML/JSON)
  • status: kubectl/HTTP API (YAML/JSON)

• Namespaces
• Labels and selectors, annotations
• Pods
• Deployments
• Services
Boundaries
Running on Kubernetes?
Operators

operator = custom resource + custom controller
A short deployment history

- Shell scripts, Make files
- Ansible, Chef, Puppet, etc.
- Helm, ksonnet, kustomize, etc.
- GitOps
Git 101
What is GitOps?
What is GitOps?

GitOps is a standardized workflow for how to deploy, configure, monitor, update and manage infrastructure-as-code.

Core idea is having a Git repository that contains declarative descriptions of the infrastructure desired in the production environment and an automated process to make production environment match the described state in the repository.
What is GitOps?

- An operation model
- Derived from operation knowledge
- Technology agnostic
- A set of principles
- A way to speed up your team
GitOps: an operation model

Providing ...

- A single source of truth for the desired system’s state
- Separation of concerns between development and deployment process
- Transparency and auditability
- Risk reduction (rollbacks)
Why should we care?

- Auditing and attribution
- Separation of concerns
- No crossing security boundary
- Process & constraints enforcement
- Great software ↔ human collaboration point
- Easy to validate for correctness (policies)
- System can self heal
The GitOps Model
Kubernetes cluster
kubectl (direct access)
configuration repository
deployment agent
state continuously monitored
control loop
GitOps Principles
1. The entire system is described declaratively.

2. The canonical desired system state is versioned (Git).

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

4. Software agents ensure correctness and alert on divergence.
The entire system is described **declaratively**

Beyond code, data ⇒

- Implementation independent
- Easy to abstract in simple ways
- Easy to validate for correctness
- Easy to generate & manipulate from code
The entire system is described *declaratively*.
The canonical desired system state is versioned (Git)

Canonical Source of Truth (DRY)

With declarative definition, trivialises rollbacks

Excellent security guarantees for auditing

Sophisticated approval processes

Great software ↔ human collaboration point
The desired system state is versioned.
Approved changes to the desired state are **automatically applied** to the system

Significant velocity gains

Privileged operators don’t cross security boundaries

Separates the **What** and the **How**
Approved changes to the desired state are automatically applied to the system.
Software agents ensure **correctness** and **alert** on divergence

Continuously checking that desired state is met

System can self-heal

Recovers from errors without intervention (layer 8 issues)

It’s the control loop for your operations
Software agents ensure correctness and alert on divergence.
Typical CI/CD pipeline

Shares credentials cross several logical security boundaries.

Continuous Integration

 Boundary

Continuous Delivery/Deployment
GitOps pipeline

Credentials are never shared across a logical security boundary.
GitOps pipeline

Credentials are never shared across a logical security boundary.

- **Git creds**
  - dev (RW)
  - code repo (RO)

- **CI creds**
  - CI (RW)
  - container registry (RO)

- **CR creds**
  - CR creds
  - operator (RW)
  - config repo

- **cluster API creds**
  - deploy (RW)

- process & constraints enforcement

---

veloCityconf.com

#VelocityConf
Example workflow

• One Git branch for development:
  • All developers can push changes there for testing purposes

• One Git branch for staging:
  • Project lead or devops manager has ability to push/merge here for testing the complete staged application

• One Git branch for production:
  • The system operations manager is the only person to have push/merge authorization here
Example workflow

Development

Developer writes code, compiles, and builds an image using the standard CI process. Developer pushes the image(s), and updates the manifest in the GitOps development branch. Developer pushes the new manifest into GitOps, and the development cluster now runs the new images.

Staging

Once the testing is complete, the developer enters a pull request from the developer GitOps to the staging GitOps. The pull request is reviewed, and if accepted, is merged into the staging GitOps. The staging cluster is updated with the new application.

Production

Once the staging and integration is complete, the DevOps manager enters a pull request from the staging GitOps to the production GitOps. The pull request is reviewed, and if accepted, is merged into the production GitOps. The production cluster is updated with the new application.
GitOps in Action!
Tooling

- CNCF Flux
- ArgoCD
- Gitkube
- Tekton
- JenkinsX
Flux

github.com/fluxcd/flux
Trainings environment:

301.sh/velocity-gitops-101

Trainings repo:

github.com/mhausenblas/gitops101
So what actually happened?

1. We installed the Flux agent in our cluster
2. We added the agent key to our repository, so that it can read and write the configuration
3. We configured the agent to watch to our repository
4. The agent noticed some manifests in the repository, and automatically applied them
5. Kubernetes deployed the manifests
Progressive delivery
What is progressive delivery?

Progressive Delivery is Continuous Delivery with fine-grained control over the blast radius, requiring:

- CI pipeline that produces immutable build artifacts
- CD pipeline designed for desired state reconciliation
- Smart routing for user facing apps and service to service communication
- Observability (performance stats + business metrics)
- Fail fast mentality
Flagger

Flagger is a Kubernetes operator that automates the promotion of canary deployments using App Mesh, Istio, Gloo or NGINX routing for traffic shifting and Prometheus metrics for canary analysis.

Flagger implements a control loop that gradually shifts traffic to the canary while measuring key performance indicators. Based on the KPIs analysis a canary is promoted or aborted.
Flagger

Deployment
Canary
HPA
Mesh

Cluster repo → Flux → Flagger → Canary → App Mesh

ECR → Prometheus
Challenges
Technical challenges

• All your artifacts in VCS such as Git?

• Tooling selection
  • app-focused vs infra-focused
  • pace of ecosystem
Organizational challenges

• Is there a devops mentality in your organization?
• Is your organization ready for “cloud native”?
Recap and Resources
Recap: GitOps CI/CD

- Git is our single source of truth
- Deployments and rollback are all done via Git
- Auditing built-in
- Having separate pipelines for CI and CD enables better security
- Automated the deployment
- It’s easier to deal with if a deployment goes wrong
Resources

- https://www.gitops.tech
- https://github.com/weaveworks/awesome-gitops
- https://www.reddit.com/r/kubernetes/comments/c2wdgz/gitops_in_production_share_your_experiences/
Rate the session

Cyberconflict: A new era of war, sabotage, and fear

9:55am-10:10am Wednesday, March 27, 2019
Location: Ballroom
Secondary topics: Security and Privacy

We’re living in a new era of constant sabotage, misinformation, and fear, in which everyone is a target, and you’re often the collateral damage in a growing conflict among states. From crippling infrastructure to sowing discord and doubt, cyber is now the weapon of choice for democracies, dictators, and terrorists.

David Sanger explains how the rise of cyberweapons has transformed geopolitics like nothing since the invention of the atomic bomb. Moving from the White House Situation Room to the dens of Chinese, Russian, North Korean, and Iranian hackers to the boardrooms of Silicon Valley, David reveals a world coming face-to-face with the perils of technological revolution—a conflict that the United States helped start when it began using cyberweapons against Iranian nuclear plants and North Korean missile launches. But now we find ourselves in a conflict we’re uncertain how to control, as our adversaries exploit vulnerabilities in our hyperconnected nation and we struggle to figure out how to deter these complex, short-of-war attacks.

David Sanger
The New York Times

David E. Sanger is the national security correspondent for the New York Times as well as a national security and political contributor for CNN and a frequent guest on CBS This Morning, Face the Nation, and many PBS shows.