

Developing, Testing and Optimizing Microservice Applications on Kubernetes – Getting Started

1



Develop a passion for
learning.

Overview



Microservice development on Kubernetes requires a specific methodology and tool set to make it easy to produce resilient self-healing applications. A good developer does not automatically make a good application on Kubernetes but with some foundational skills it is possible to greatly improve development velocity and the quality of the end product. Students will learn about application development, debugging, and testing with Kubernetes. Monitoring, tracing, and service mesh implementations with Kubernetes.

LOGISTICS



Class Hours:

- Start time is X:XXam
- End time is X:XXpm
- Class times may vary slightly for specific classes
- Breaks mid-morning and afternoon (15 minutes)



Lunch:

- Lunch is 11:45am to 1pm
- Yes, 1 hour and 15 minutes
- Extra time for email, phone calls, or simply a walk.



Telecommunication:

- Turn off or set electronic devices to vibrate
- Reading or attending to devices can be distracting to other students
- Try to delay until breaks or after class

Miscellaneous

- Courseware
- Bathroom
- Fire drills

DAY 1

Agility



Framing Our Journey

- Kubernetes
- Architecture
- Managing the Chaos: Labels
- Label Recommendations
- Managing the Chaos: Architecture
- Microservices
- Enterprise Agility
- Persistent Storage
- Local Development, Test, and Debug
 - What is the objective for? Local
- Dev
 - What about Local Clusters
 - What is KIND
 - KIND Security
 - Installing Kind
 - First KIND
 - KIND Node Image
 - Building your Node Image
 - Creating a Cluster
 - Testing Kubernetes with KIND
 - KIND Known Issues
 - KIND Configuration
- KIND Ingress
 - Creating a Kubernetes Ingress
- KIND Local Registry
 - What are Air-gapped
- Environments
 - KIND Working Offline
 - KIND Resources

k8s

Horizontally

Scalable Components

Debugging & Introspection
Debugging Side by Side with KIND and

What about all these K3's
What is k3s
K3s Process Architecture
Why J3s?
K3s Single Server
Node Registration
Where is k3s Going?
What is K3d
K3d CNCF Landscape
K3d Installation
Where can I Find k3d
K3d Code Home
K3d Command Tree
Deploying K3s with k3d
What is K3c
K3c Demo
K3s & K3c Integration
Debugging Local Containers
Scalability
Vertically Scalable Components
Horizontally Scalable Components
Building Applications Using Layers of

DAY 2

Velocity



- Remote Development, Test, and Debug
 - Using Telepresence
 - Attaching a Debugger to a Remote Server
 - Kubernetes Does Everything
 - Problem Statement
 - Package Manager
 - What is Helm?
 - Installing Helm
 - Helm Architecture
 - Helm Features
 - Helm Charts
 - Helm Chart Structure
 - Installing Your First Helm Chart
 - Helm Chart.yaml
 - Helm Commands
 - Understanding WordPress Application
 - Finding a WordPress Chart
 - Viewing a WordPress Chart
 - Helm Dependencies
 - Helm Repositories
 - Helm Templates
 - Linting Helm Charts and Templates
 - Debugging Templates
 - Node Namespaces
 - What is KPT?
 - KPT Architecture
 - KPT Conventions
 - KPT Packaging
 - KPT Functions
 - Using Helm and KPT
 - KPT Installation Source

- Controller/ Operator Patterns/ Creation
 - What is a Controller?
 - Controller Pattern
 - Controller in Kubernetes
 - Example: Admission Controller
 - Controller Components
 - Creating Custom Controllers
 - Controller Loop
 - What is an Operator?
 - Developing Operators
 - Building a Custom Resource Definition
 - Application patterns
 - Foundational Application Patterns
 - Behavioral Application Patterns
 - Structural Application Patterns

DAY 3

Observability



Monitoring

- Observability & Monitoring
- Observability
- Rise of Prometheus
- Installing Prometheus
- Interacting with Prometheus
- Incorporating Kube-state-metrics
- Triggering Alerts with AlertManager
- Prometheus Operator
- Exposing Metrics with Prometheus
- Deploying Prometheus Using Helm
- Node Exporter
- Grafana
- Grafana Dashboard
- Customer Metrics Trigger
- Logging Solution
- Loki

Service Mesh

- What is Service Mesh?
- Why Service Mesh?
- Optimizing Communication
- Kubernetes and Service Mesh
- Implementations
- Canary
- Problem Statement
- Architecture
- Best Practices
- Implementing Retry Logic
- Rate Limiter & Load Shedders
- Implementing Weighted Routing
- Implementing Access Controls

Tracing

- What is Distributed Tracing?
 - What's in a Name?
 - Tracing at the Code Level
 - Trace Spans
 - Tracing Information
 - Application Performance Monitoring
 - Digital Performance Monitoring
 - DPM Correlation
 - AI Ops
 - Manual Code Tracing
 - OpenTracing Concepts
 - Tracing Patterns
 - Open Telemetry
 - Open Telemetry with Jaeger
 - Tracing Continuity
 - Trace Storage Backends
 - Collecting Telemetry from Mesh
 - Implementing Tracing at Service Mesh Level
 - Jaeger
 - Installing Jaeger
- References

Meet The Instructor

George Niece AKA geo

Digital Transformation, DevSecOps, IoT Consultant with a Software Engineering, Digital Commerce, and IT operations background. Focused on cloud-native application modernization, datacenter divestiture, and cloud adoption.



Twitter
[@georgeniece](https://twitter.com/georgeniece)

LinkedIn
[Linkedin.com/in/GeorgeNiece](https://www.linkedin.com/in/GeorgeNiece)

mail
George.Niece@DigitalTransformationStrategies.net

Expertise

- Cloud
- XaaS
- AppDev
- IoT
- Automation
- CI/CD
- Microservices
- Agile

Framing our Journey



Kubernetes

The name Kubernetes originates from Greek, meaning helmsman or pilot. Google open-sourced the Kubernetes project in 2014. Kubernetes combines **over 15 years of Google's experience** running production workloads at scale with best-of-breed ideas and practices from the community.

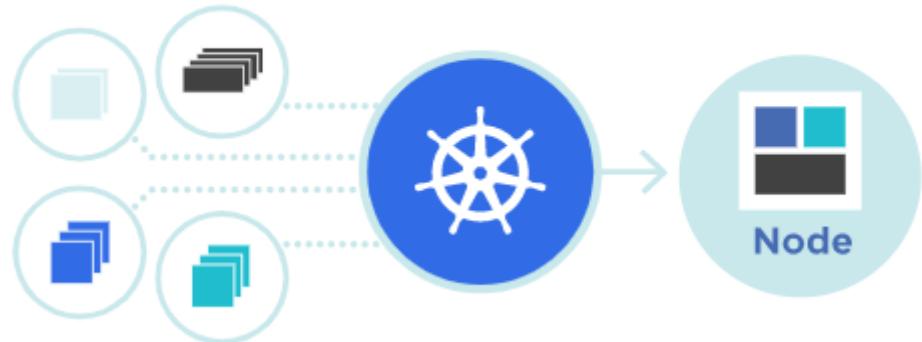


Jenga for Kubernetes

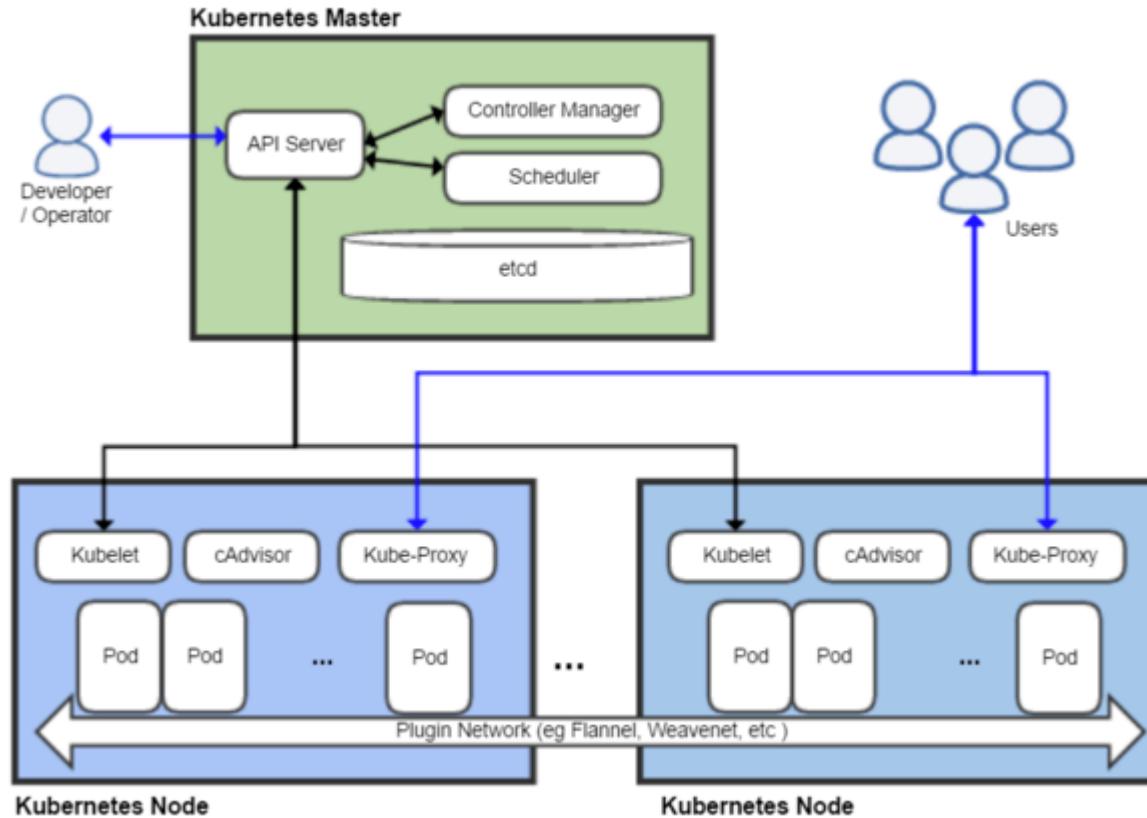


Terms and Definitions

- Pod
- Service
- Volume
- Namespace
- ReplicaSet (RS)
- Deployment
- StatefulSet
- DaemonSet
- Job



Architecture



Managing the Chaos : Labels

I Labels



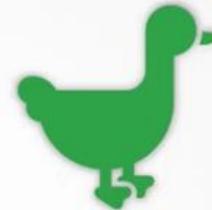
Class	Mammal
Kind	Domestic
Color	Green



Class	Reptile
Kind	Wild
Color	Purple



Class	Mammal
Kind	Domestic
Color	Orange



Class	Bird
Kind	Domestic
Color	Green

Managing the Chaos : Label Selectors

ISelectors



Class	Mammal
Kind	Domestic
Color	Green

Class = Mammal

&

Color = Green

Label Recommendations

Label Example Key	Description	Label Example value
Application-ID/Application-name	The name of the application or its ID	my-awesome-app/app-nr-2345
Version-nr	The version number	ver-0.9
Owner	The team or individual the object belongs to	Team-kube/josh
Stage/Phase	The stage or phase	Dev, staging, QA, Canary, Production
Release-nr	The release number	release-nr-2.0.1

Label Recommendations

Label Example Key	Description	Label Example value
Tier	Which tier the app belongs to	front-end/back-end
Customer-facing	Is the resource part of an app that is customer facing?	Yes/No
App-role	What roles does the app have	Cache/Web/Database/Auth
Project-ID	The associated project ID	my-project-276
Customer-ID	The customer ID for the resource	customer-id-29

Managing the Chaos : Annotations

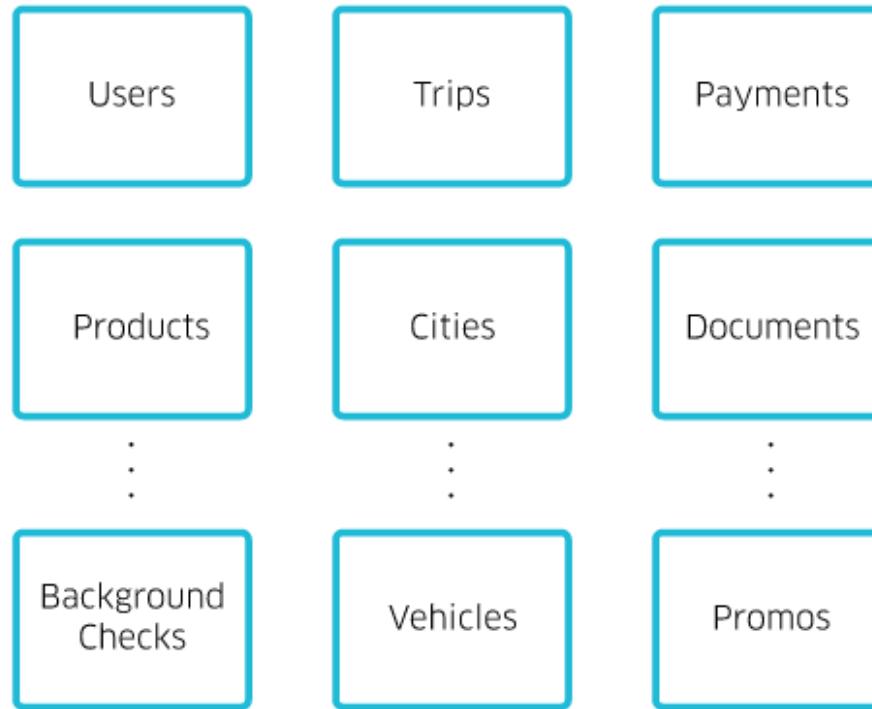
```
apiVersion: apps/v1
kind: ReplicaSet
metadata:
  name: webapp-rs
  labels:
    app: webapp
  annotations:
    buildversion: 1.34

spec:
  template:
    metadata:
      name: webapp-pods
      labels:
        app: webapp
        tier: front-end
    spec:
      containers:
        - name: httpd
          image: httpd
          imagePullPolicy: IfNotPresent
          ports:
            - containerPort: 80
```

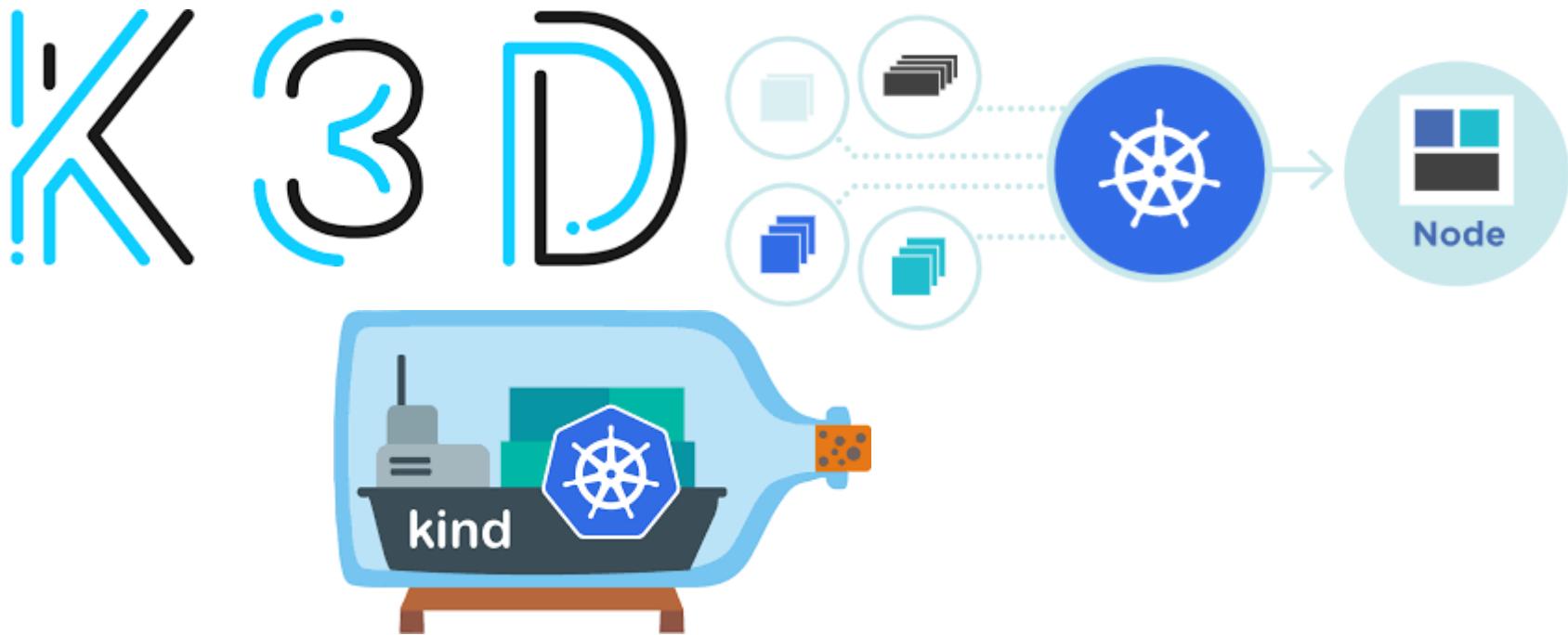
Microservices

- Services in a **microservice architecture (MSA)** communicate over a **network** using technology-agnostic **protocols** such as HTTP.
- **Organized around business capabilities.**
- **Implemented** using different **programming languages**, **databases**, and components, **depending on what fits best**.
- **Small in size**, messaging-enabled, bounded by contexts, **decentralized** and deployed **with automated processes**.

Enterprise Agility

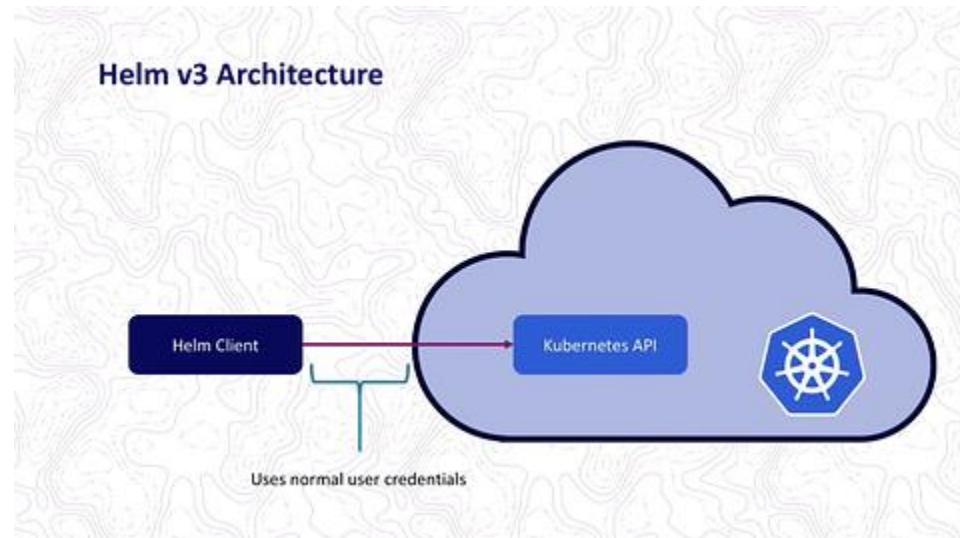


Local development



Helm3

Kubernetes follows a depreciation policy where as a new API version or format become available the old one won't be supported anymore. This is the case with the transition from Helm2 to Helm3



Observability



JAEGER



Grafana loki



Grafana



Prometheus

Monitoring

POP QUIZ:

Kubernetes: Framing Our Journey



What is Kubernetes?

- A: a set of platform as a service products
- B: An open-source container orchestration tool
- C: a tool that streamlines installing and managing applications

POP QUIZ:

Kubernetes: Framing Our Journey



What is Kubernetes?

A: a set of platform as a service products

B: An open-source container orchestration tool

C: a tool that streamlines installing and managing applications

POP QUIZ:

Kubernetes: Framing Our Journey



What is K8s?

- A: A term for Kubernetes
- B: A Kubernetes Service
- C: A term for a local cluster



POP QUIZ:

Kubernetes: Framing Our Journey



What is K8s?

- A: A term for Kubernetes
- B: A Kubernetes Service
- C: A term for a local cluster



POP QUIZ:

Kubernetes: Framing Our Journey



What is a node in Kubernetes?

- A: high-level structures that wrap one or more containers
- B: the smallest fundamental unit of computing hardware.
- C: a performance monitoring and metrics collection system

POP QUIZ:

Kubernetes: Framing Our Journey



What is a node in Kubernetes?

- A: high-level structures that wrap one or more containers
- B: the smallest fundamental unit of computing hardware.
- C: a performance monitoring and metrics collection system

Experiment - Foundation



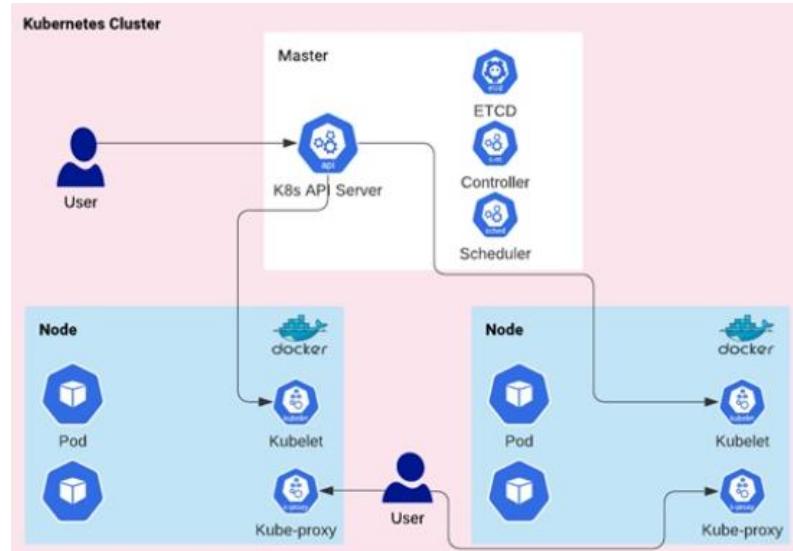
Local Development, Testing, and Debugging with **KIND**



What is the objective for? Local Dev

[kind](#) is a tool for running local Kubernetes clusters using Docker container “nodes”.

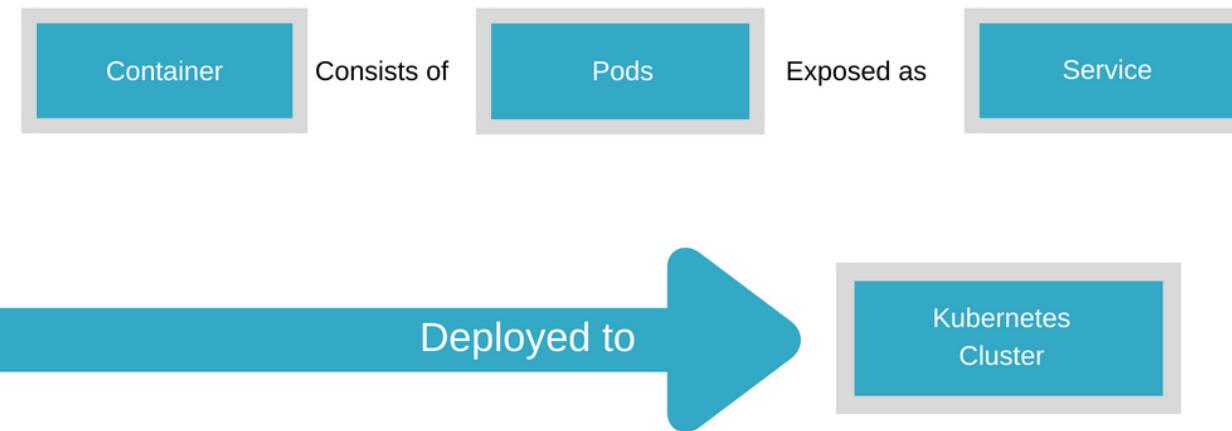
kind was primarily designed for testing Kubernetes itself, but may be used for local development or CI



What about Local Clusters



How would a local cluster empower development. We'll look into a number of ways



What is KIND?

KIND stands for Kubernetes IN Docker, and as the name suggests, it creates a Kubernetes cluster using Docker to host the nodes. This is a novel approach, that takes advantage of Docker's easy, self-contained deployments and cleanup to create the test Kubernetes infrastructure.

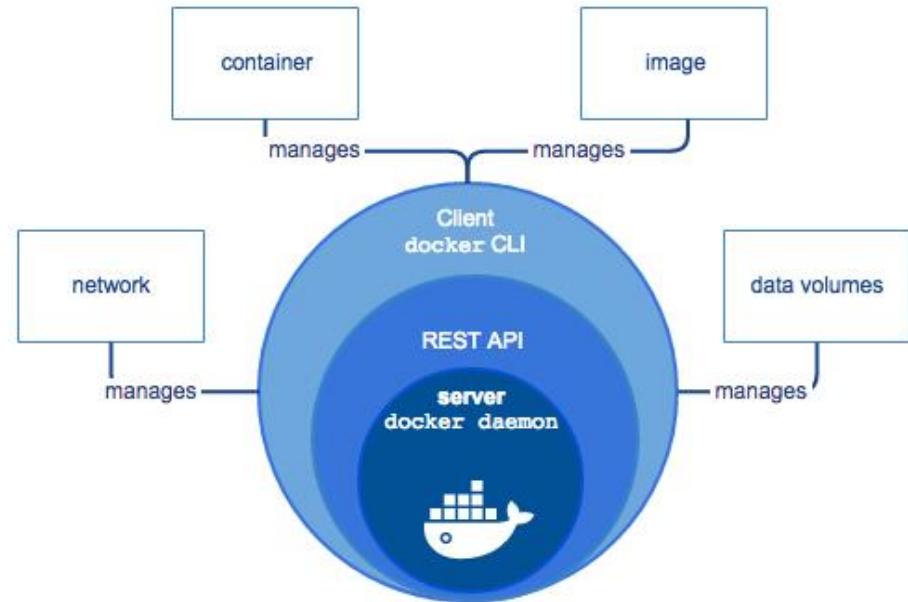
[kind](#) is a tool for running local Kubernetes clusters using Docker container “nodes”.
kind was primarily designed for testing Kubernetes itself, but may be used for local development or CI



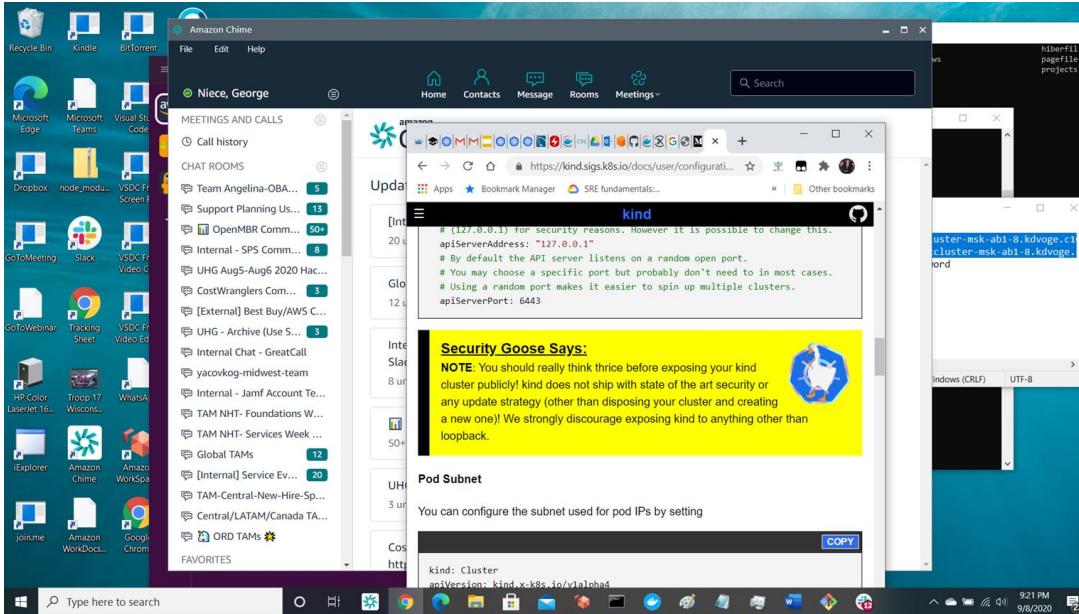
What is Docker?

Docker is a set of platform as a service products that use OS-level virtualization to deliver software in packages called containers.

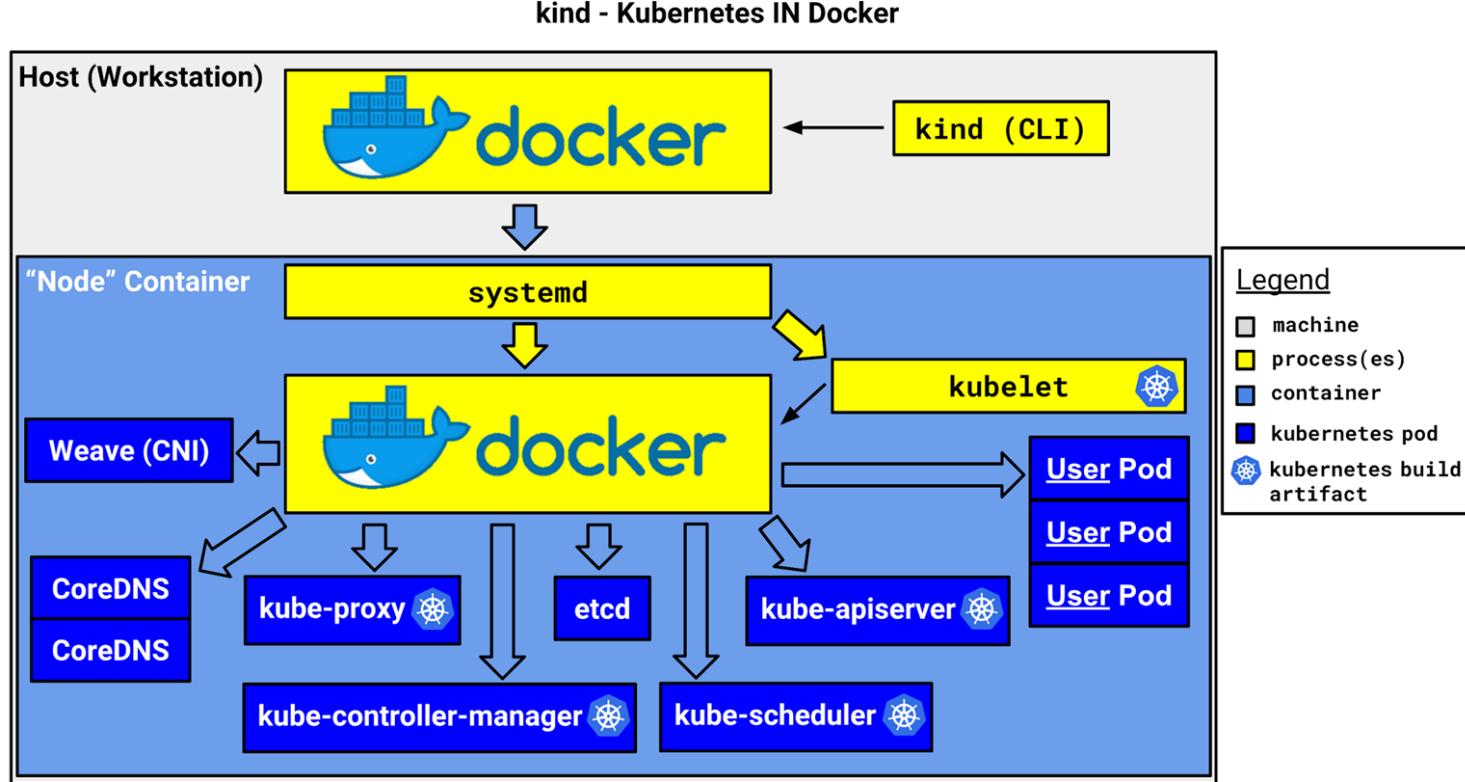
Containers are isolated from one another and bundle their own software, libraries and configuration files; they can communicate with each other through well-defined channels.



KIND Security



KIND



KIND ...

Being “production workload ready” - kind is meant to be used:

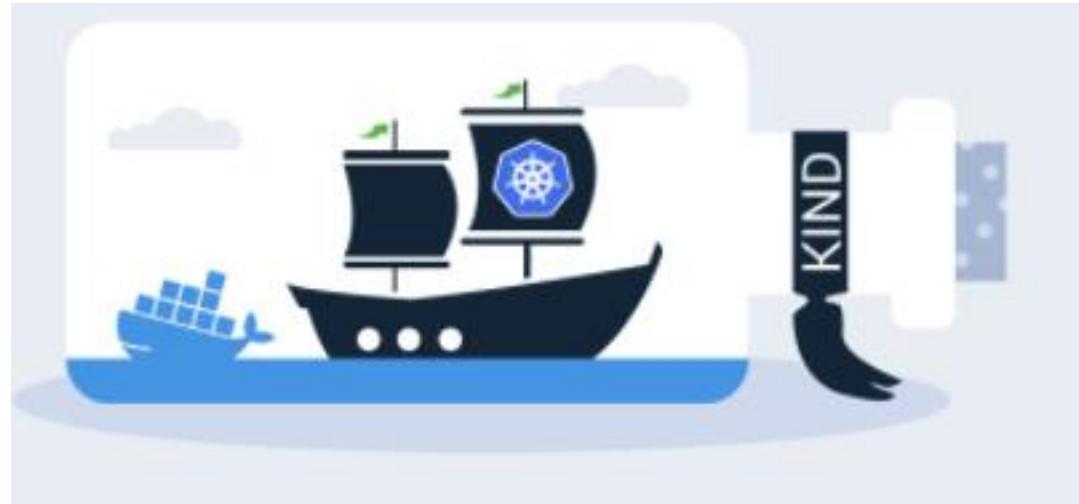
- for testing Kubernetes itself
- for testing against Kubernetes (EG in CI on Travis, Circle, etc.)
- for “local” clusters on developer machines
- **NOT** to host workloads serving user traffic etc.



Installing KIND

KIND requires Go and Docker

Specifically, Go 1.14 or newer, and preferably 1.15



First KiND

```
$ time kind create cluster
Creating cluster "kind" ...
✓ Ensuring node image (kindest/node:v1.16.3) 
✓ Preparing nodes 
✓ Writing configuration 
✓ Starting control-plane 
✓ Installing CNI 
✓ Installing StorageClass 
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Not sure what to do next? 😊 Check out https://kind.sigs.k8s.io/docs/user/quick-start/

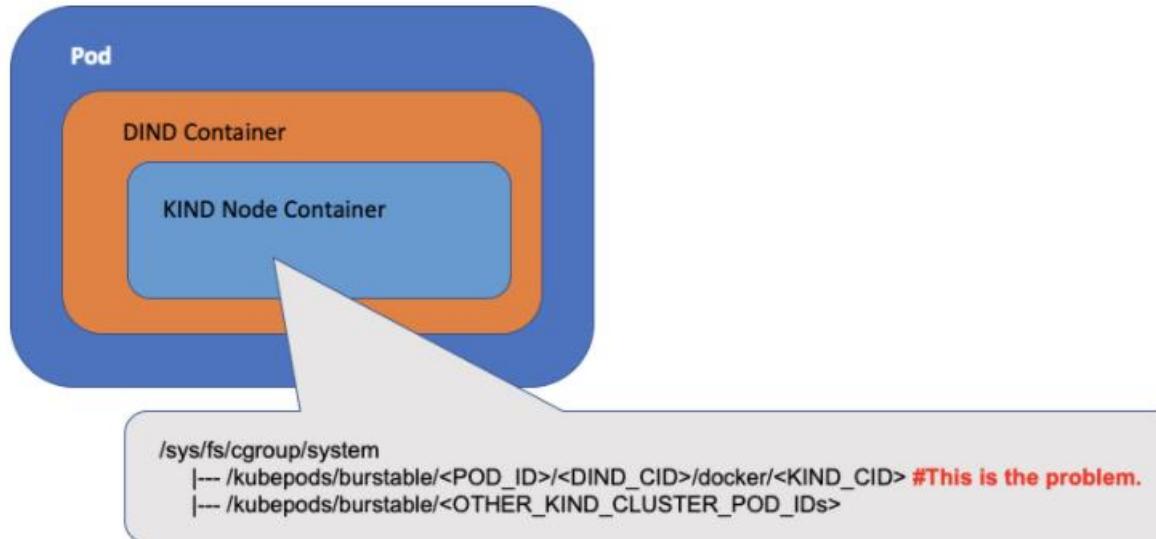
real      0m21.890s
user      0m1.278s
sys       0m0.790s
```

Experiment – Working with Kind



Kind Node Image

Kind bootstraps a Kubernetes cluster using a pre-built [node image](#) - you can find it on docker hub [kindest/node](#). If you desire you can build the node image yourself. To specify another image use the --image flag.



Building your Node Image

The “[node](#)” image is a Docker image for running nested containers, systemd, and Kubernetes components.

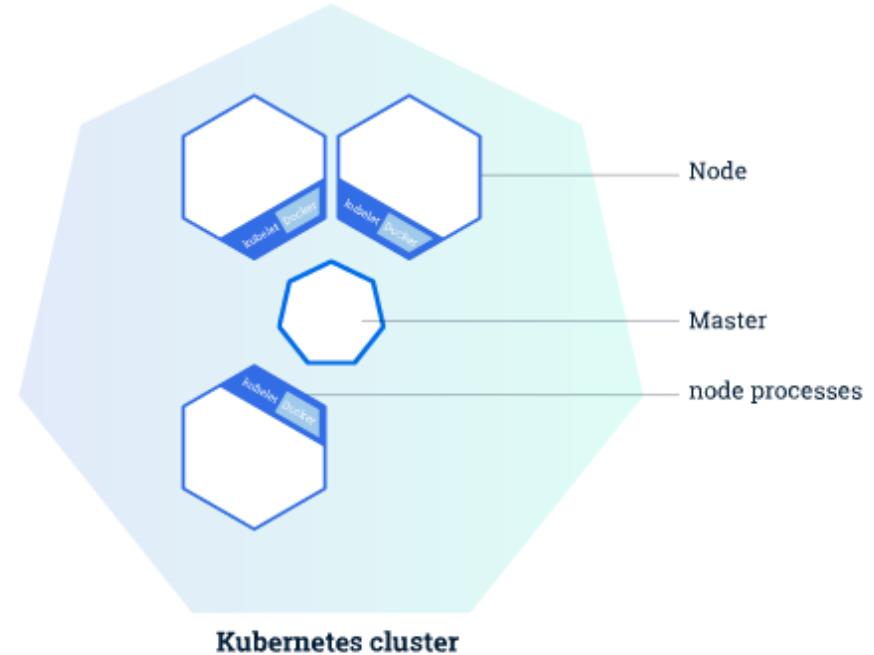
This image is built on top of the “[base](#)” image.



Creating a Cluster

Creating a Kubernetes cluster is as simple as `kind create cluster`.

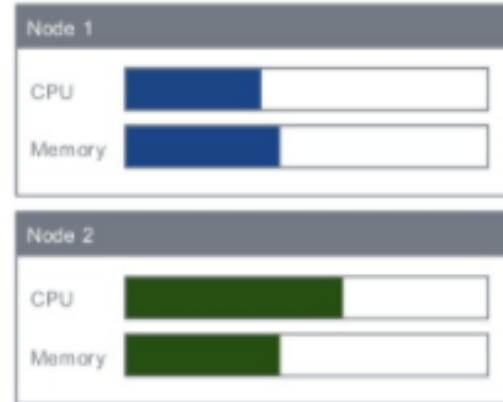
By default, the cluster will be given the name `kind`. Use the `--name` flag to assign the cluster a different context name.



Kind Known Issues



POD SCHEDULING



Kind Configuration

A minimal valid config is:

```
kind: Cluster
apiVersion: kind.x-
k8s.io/v1alpha4
```

```
$ time kind create cluster
Creating cluster "kind" ...
✓ Ensuring node image (kindest/node:v1.16.3) [!]
✓ Preparing nodes [!]
✓ Writing configuration [!]
✓ Starting control-plane [!]
✓ Installing CNI [!]
✓ Installing StorageClass [!]
Set kubectl context to "kind-kind"
You can now use your cluster with:

kubectl cluster-info --context kind-kind

Not sure what to do next? 😊 Check out https://kind.sigs.k8s.io/docs/user/quick-start/

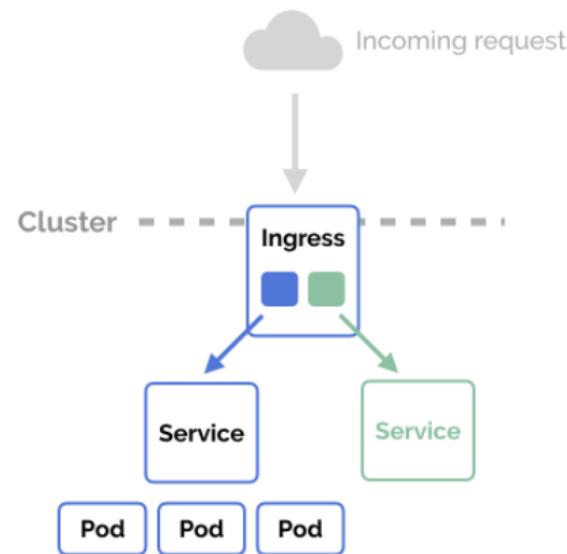
real    0m21.890s
user    0m1.278s
sys     0m0.790s
```

Kind Ingress

Setting Up An Ingress Controller

We can leverage KIND's `extraPortMapping` config option when creating a cluster to forward ports from the host to an ingress controller running on a node.

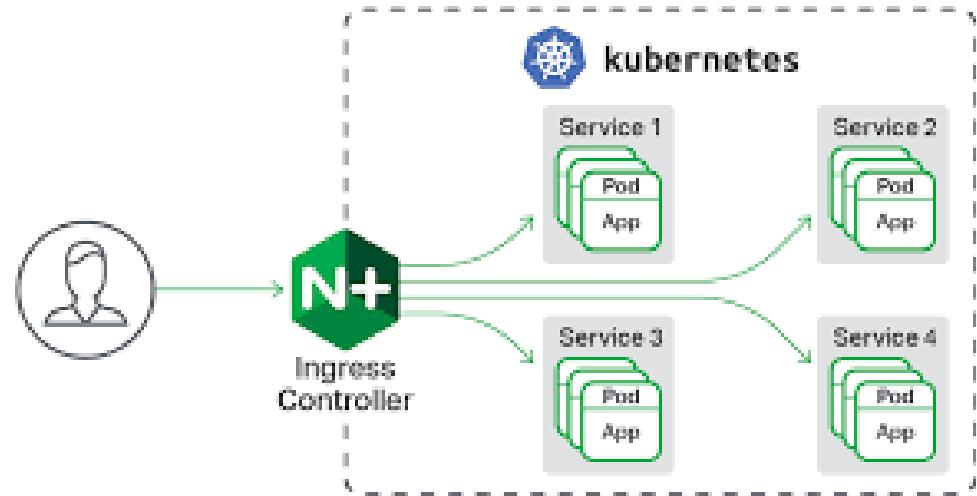
We can also setup a custom node label by using `node-labels` in the `kubeadm InitConfiguration`, to be used by the ingress controller `nodeSelector`.



Ingress

Ingress exposes HTTP and HTTPS routes from outside the cluster to services within the cluster. Traffic routing is controlled by rules defined on the Ingress resource.

```
internet
  |
  [ Ingress ]
  --|-----|
  [ Services ]
```

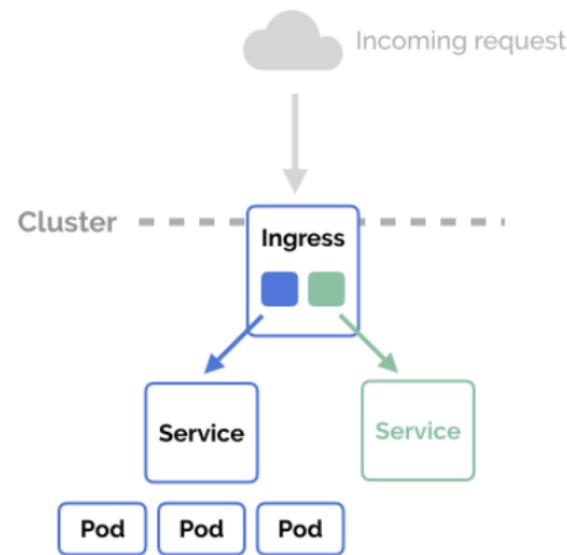


Creating a Kubernetes Ingress

Setting Up An Ingress Controller

We can leverage KIND's `extraPortMapping` config option when creating a cluster to forward ports from the host to an ingress controller running on a node.

We can also setup a custom node label by using `node-labels` in the `kubeadm InitConfiguration`, to be used by the ingress controller `nodeSelector`.



Kind Local Registry

examples/kind-with-registry.sh COPY

```
#!/bin/sh
set -o errexit

# create registry container unless it already exists
reg_name='kind-registry'
reg_port='5000'
running="$(docker inspect -f '{{.State.Running}}' "${reg_name}" 2>/dev/null || true)"
if [ "${running}" != 'true' ]; then
  docker run \
    -d --restart=always -p "${reg_port}:5000" --name "${reg_name}" \
    registry:2
fi

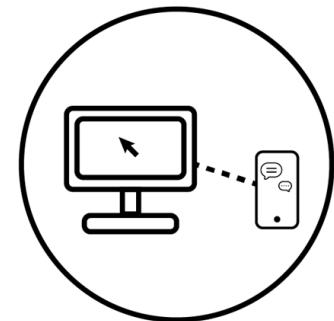
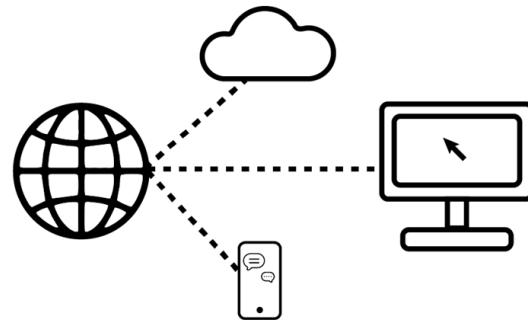
# create a cluster with the local registry enabled in containerd
cat <<EOF | kind create cluster --config=-
kind: Cluster
apiVersion: kind.x-k8s.io/v1alpha4
containerdConfigPatches:
- |-
  [plugins."io.containerd.grpc.v1.cri".registry.mirrors."localhost:${reg_port}"]
  endpoint = ["http://${reg_name}:${reg_port}"]
EOF

# connect the registry to the cluster network
docker network connect "kind" "${reg_name}"

# tell https://tilt.dev to use the registry
# https://docs.tilt.dev/choosing_clusters.html#discovering-the-registry
for node in $(kind get nodes); do
  kubectl annotate node "${node}" "kind.x-k8s.io/registry=localhost:${reg_port}";
done
```

What are air-gapped environments?

air gapping is a network security measure employed on one or more computers to ensure that a secure computer network is physically isolated from unsecured networks, such as the public Internet or an unsecured local area network.



Air-gapped Network

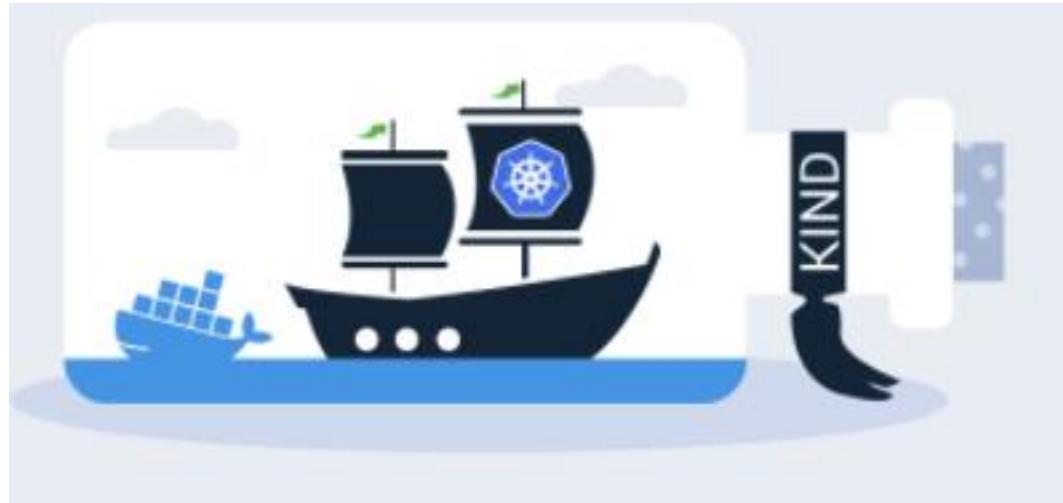
Devices included in the air-gapped network are physically isolated and can communicate with each other, but cannot communicate with any other network outside of the air-gap.

Kind Working Offline

Some users may work in an offline environment, let's talk about how to create a cluster using kind in this environment.



Testing Kubernetes with KiND



Types of Test

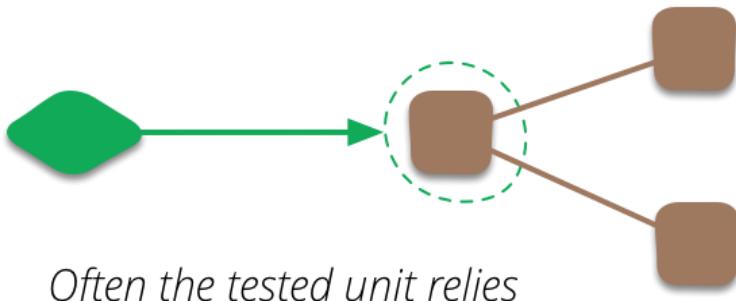
Unit Tests

- Typically make use of the “fake client”
- In-memory implementation of an apiserver/ client
- Great for simple things
 - Majority of the api-server’s functionality does not exist
 - Race and timing issues not surfaced
 - No other controllers running



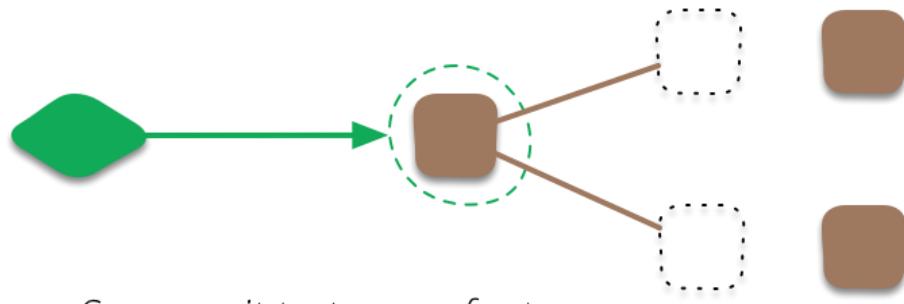
Unit Tests

Sociable Tests



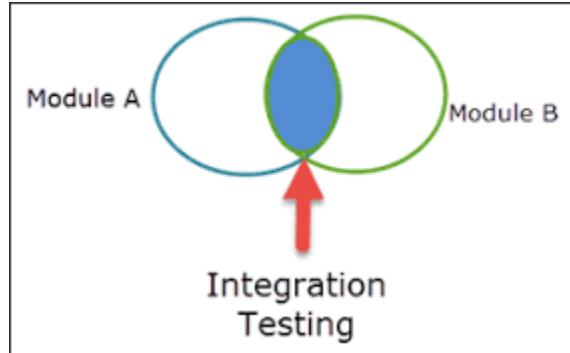
*Often the tested unit relies
on other units to fulfill its
behavior*

Solitary Tests



*Some unit testers prefer to
isolate the tested unit*

Types of Test



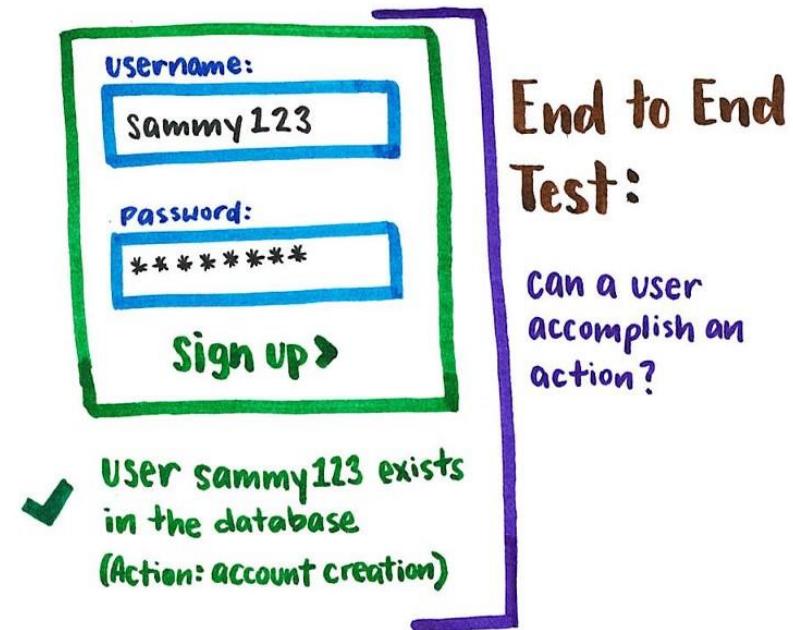
Integration tests

- Kubebuilder / controller-runtime use this approach a lot
- Run etcd + apiserver (+ optionally controller-manager)
- Why
 - Admission control
 - Timing issues

Types of Test

End-to-end tests

- Start our full Kubernetes cluster(s)
- Gives the ultimate flexibility
- Black box testing (we don't assume implementation)
- Certain edge cases are only picked up in a real environment
- Kubernetes has a lot of controllers
 - This can be slow and/or expensive



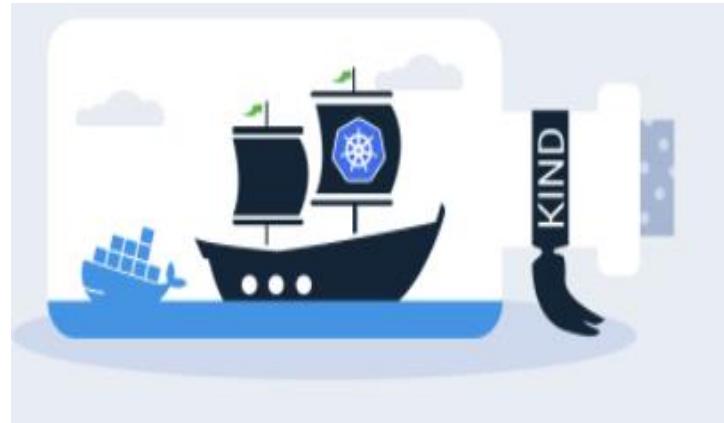
Manipulating the test environment

- Sometimes, custom configuration is required
- KIND exposes the full Kubeadm config surface
- Running in a container makes it easier to create portable environments:
 - Changing the service CIDR
 - Enabling alpha level features
 - Pretty much anything else



KIND as a Testing Tool

- Boot cluster “ kind create cluster”
- Build your application
- Sideload application images “kind load docker-image myapp:latest
- {run tests} “ KUBECONFIG=\$(kind get kubeconfig-path)./run_tests.sh
- Repeat!



Passing additional Config

```
1 # this config file is similar to the default, except we set the cluster's
2 # service cidr range to be 10.0.0.0/16.
3 # we do this because we need a fixed/predictable clusterIP of 10.0.0.15 for the
4 # nginx-ingress service, in order to perform HTTP01 validations during tests.
5
6 apiVersion: kind.sigs.k8s.io/v1alpha2
7 kind: Config
8 nodes:
9 - role: control-plane
10   kubeADMConfigPatches:
11   - |
12     # config generated by kind
13     apiVersion: kubeADM.k8s.io/v1beta1
14     kind: ClusterConfiguration
15     metadata:
16       name: config
17     networking:
18       serviceSubnet: 10.0.0.0/16
```

Advanced Networking Config

Using a custom CNI

```
1 # Boot the cluster without installing a CNI
2 kind: Cluster
3 apiVersion: kind.sigs.k8s.io/v1alpha3
4 networking:
5   disableDefaultCNI: true
6
```

Kind Resources

KubeCon Talks

Deep Dive: KIND - Benjamin Elder & Antonio Ojea

Using KIND in CI

How to use KIND with MetallLB

<https://maulion.dev/posts/kind-metallb/>

How to Test a Kubernetes PR with KIND

<https://maulion.dev/posts/kind-k8s-testing/>

Using Contour Ingress with KIND

<https://projectcontour.io/kindly-running-contour/>

POP QUIZ:

Kubernetes: Local Development, Test, and Debug



What does KIND stand for?

- A: Kube
- B: Kubernetes in Docker
- C: It is not an acronym



POP QUIZ:

Kubernetes: Local Development, Test, and Debug



What does KIND stand for?

A:

B: Kubernetes in Docker

C: It is not an acronym

POP QUIZ:



Kubernetes: Local Development, Test, and Debug



How are Kubernetes and Docker Related?

- A: Docker can't deploy automatic rollbacks; Kubernetes can deploy rolling updates as well as automatic rollbacks
- B: Docker allows for auto-scaling when using Kubernetes
- C: Kubernetes allows for the manual linking and orchestration of several containers, running on multiple hosts that have been created using Docker.

POP QUIZ:



Kubernetes: Local Development, Test, and Debug



How are Kubernetes and Docker Related?

A: Docker can't deploy automatic rollbacks; Kubernetes can deploy rolling updates as well as automatic rollbacks

B: Docker allows for auto-scaling when using Kubernetes

C: Kubernetes allows for the manual linking and orchestration of several containers, running on multiple hosts that have been created using Docker.

POP QUIZ:

Kubernetes: Local Development, Test, and Debug



What is a Local Cluster?

- A:
- B:
- C:



Experiment – Air Gapped Kind



Chaos Engineering

Concepts



Why do we need Chaos Engineering?



Chaos Engineering is the foundation of a new application paradigm in resiliency and reliability. Rather than inject bugs into our applications, we test them in their natural state injecting faults.

Why does the Business need Chaos Engineering?



All of these companies suffered catastrophic downtime in that last year that cost millions of dollars in lost revenue and significant reputational damage

Building Confidence in Complicated and Complex Systems

When you create a microservice-based application, you need to deal with complexity. Of course, a single microservice is simple to deal with, but dozens or hundreds of types and thousands of instances of microservices is a complex problem. It isn't just about building your microservice architecture—you also need high availability, addressability, resiliency, health, and diagnostics if you intend to have a stable and cohesive system.





Synoptic Illegibility

If you can't write down exactly what **really** happens, (can't write a synopsis) you can't automate an ad-hoc and messy system.

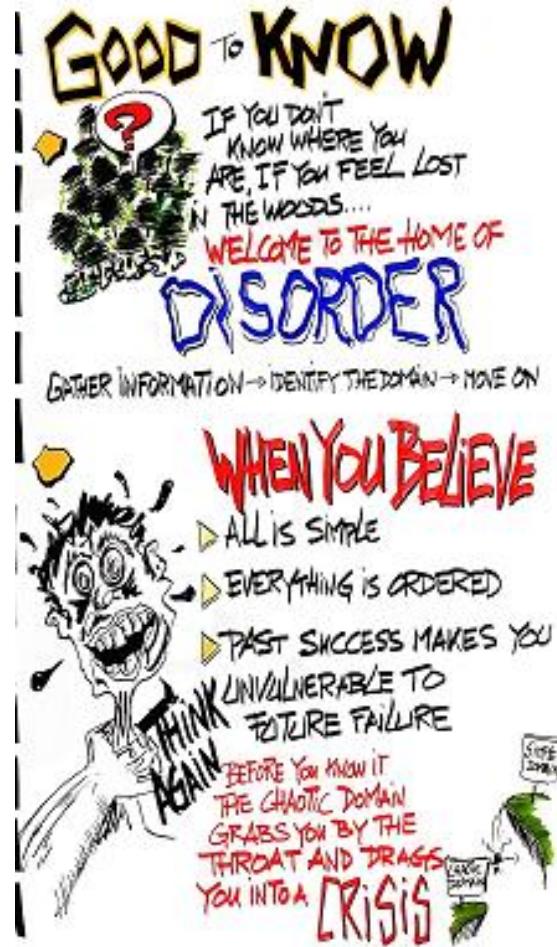


*The journey of a thousand
miles begins with a single
step.*

-Lao Tzu

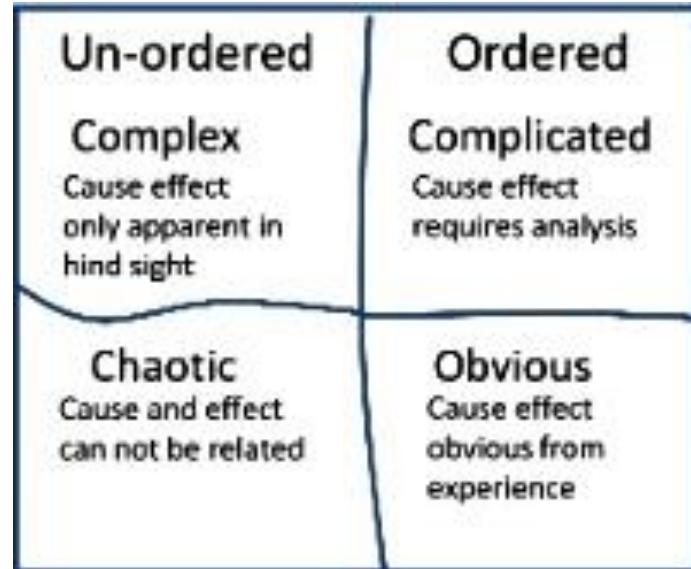
What is Cynefin?

kuh-NEV-in is the Welsh word for habitat.
Developed in the 2000's by IBM as a way to put perspective on complex systems



Understanding and Working with the Emergent and Novel with Cynefin, Chaos and Microservices

We need to acknowledge that systems are not always stable and the state of the system may change over time. A system may be stable one day and we believe we know how it works. However we may find that its performance is degrading overtime or it may simply break.

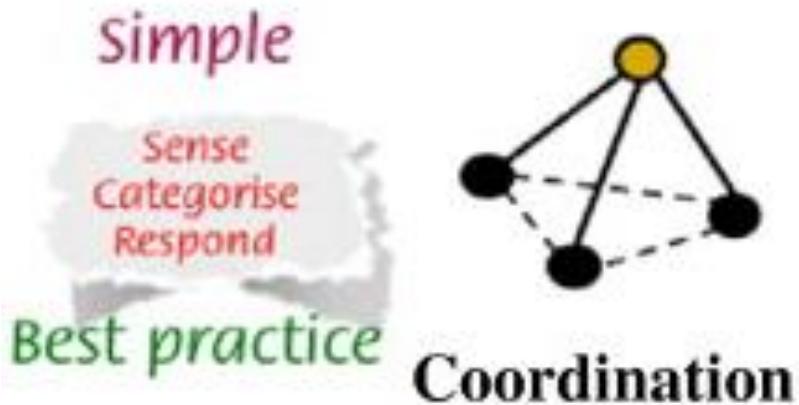


Ordered and unordered chaos

In the ordered, we may have to involve an expert (say a mechanic in the case of a car) to analyze what is going on. In the case of the un-ordered, where a car has broken down and we have been thrown into chaos, we may need to involve a rescue service to recover the vehicle and take it to a garage in order for a mechanic to undertake the analysis.

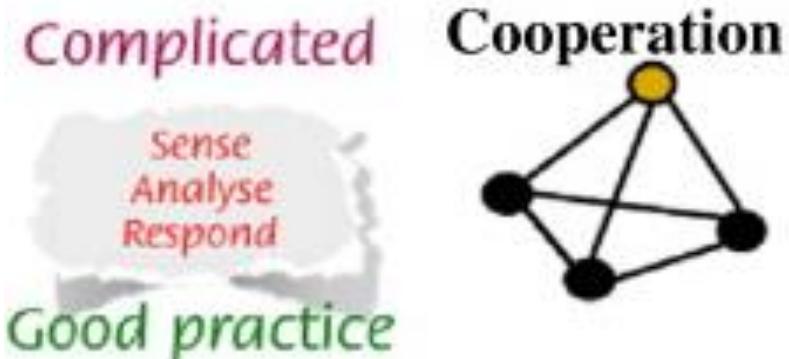


Domains of chaos – SIMPLE or Obvious



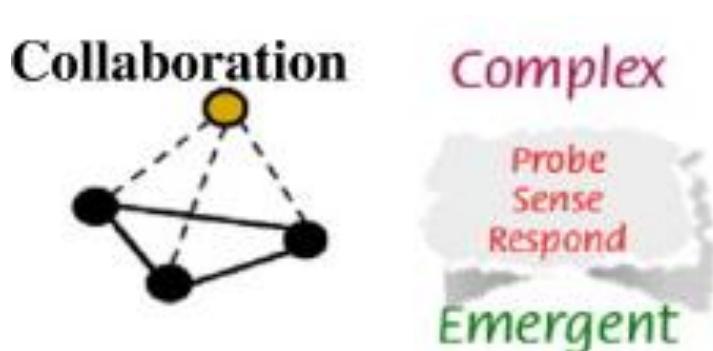
Obvious (known knowns) – here we know what we are doing and have seen it a thousand times before, so we sense, categorize and respond (S-C-R). Here we expect to see best practices employed.

Domains of chaos - complicated



Complicated (known unknowns) – we don't know what is going on but know that we can analyse what has happened and work it out, so we are sensing, analyzing and responding (S-A-R). This is the domain of good practice.

Domains of chaos – complex



Complex (unknown unknowns) - we cannot determine what will cause a particular outcome but we can run some experiments to see if they move us in the right direction so we probe, sense and respond (P-S-R). There is no right or wrong answer, so we may want to run a series of experiments or in fact run a number in parallel.

Domains of chaos - disorder



Disorder (not determined) – these are the items that we are yet to determine in which domain they belong.

Working with Experiments and Chaos in Practice

Chaos Engineering is a disciplined approach to identifying failures before they become outages. By proactively testing how a system responds under stress, you can identify and fix failures before they become all hands meetings and newsworthy.

Chaos Engineering lets you compare what you think will happen to what actually happens in your systems. You literally experiment (break things on purpose) to learn how to build more resilient systems.



What are Chaos Experiments?

- Experiments are documented tasks that reveal system vulnerability when performed, with the chaos engineering goal of teaching us something new.
- Usually one of two outcomes
 - Success – The system is confirmed not to be vulnerable to the experiment performed. Celebrate.
 - Success – The system is confirmed to be vulnerable to the experiment performed. Celebration may be appropriate here, as well, after all you just identified a vulnerability. After the celebration, fix the issue and run the experiment to validate.

What are Chaos Probes?

- Points of failure. Centralized systems are easier to manage but have a single point of failure.
- Fault / Tolerance. Decentralized are resilient because of the redundancy of effort.
- Scalability. Centralized - limited scalability. Distributed - more scalable. Decentralized - infinite scalability.
- Development. Centralized is simple and technically easier to develop. Decentralized requires more considerations and more technically complex.
- Security. Centralized is less secure with a single point of attack. Decentralized, especially emerging technology like chaos engineering and AI/ML, become more secure as they scale.

Why do Chaos in Production?

Systems behave differently depending on environment and traffic patterns. Since the behavior of utilization can change at any time, sampling real traffic is the only way to reliably capture the request path. To guarantee both authenticity of the way in which the system is exercised and relevance to the current deployed system, Chaos strongly prefers to experiment directly on production traffic.



POP QUIZ: Chaos engineering



- What is the biggest roadblock to chaos in most organizations?



2 MINUTE

POP QUIZ: Chaos engineering



- What is the biggest roadblock to chaos in most organizations?
- CULTURE
- No time for experiments
- Teams already too busy fixing things
- Politics
- Deeply invested in product technical roadmap (micro-services) that chaos engineering tests show are not as resilient as predicted
- Could start broad and deep conversations

2 MINUTE

Experiment- Setting up the Chaos Toolkit



Setting up the Chaos Toolkit

The Chaos Toolkit is a free, open source project that enables you to create and apply Chaos Experiments to discover, and eventually improve and address, weaknesses across your system's infrastructure, platform and application levels.

This tutorial takes you through the setup of the Chaos Toolkit's **chaos** command. It also teaches you how Chaos Toolkit Extensions, that select which layers and technologies can be targeted by your experiments, can be installed into your environment.

<https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/chaostoolkit-install>

Katacoda

For our Chaos Experiments you'll need an account. That's simple, either use an existing account on a number of providers or sign up with an email

<https://katacoda.com/>

Experiment - Our First Chaos



Our First Chaos Experiment

In this scenario you will learn how to write a simple Chaos Toolkit experiment and then use the chaos run command to execute your experiment.

<https://katacoda.com/chaostoolkit/courses/01-chaostoolkit-getting-started/simple-experiment>

Local Development, Testing, and Debugging with K3X



What about all these K3's

1. K3s - Lightweight Kubernetes Distribution
2. K3c - Container runtime for k3s
3. K3d - Fast spin-up of local k3s clusters



What is k3s

Lightweight Kubernetes Distribution



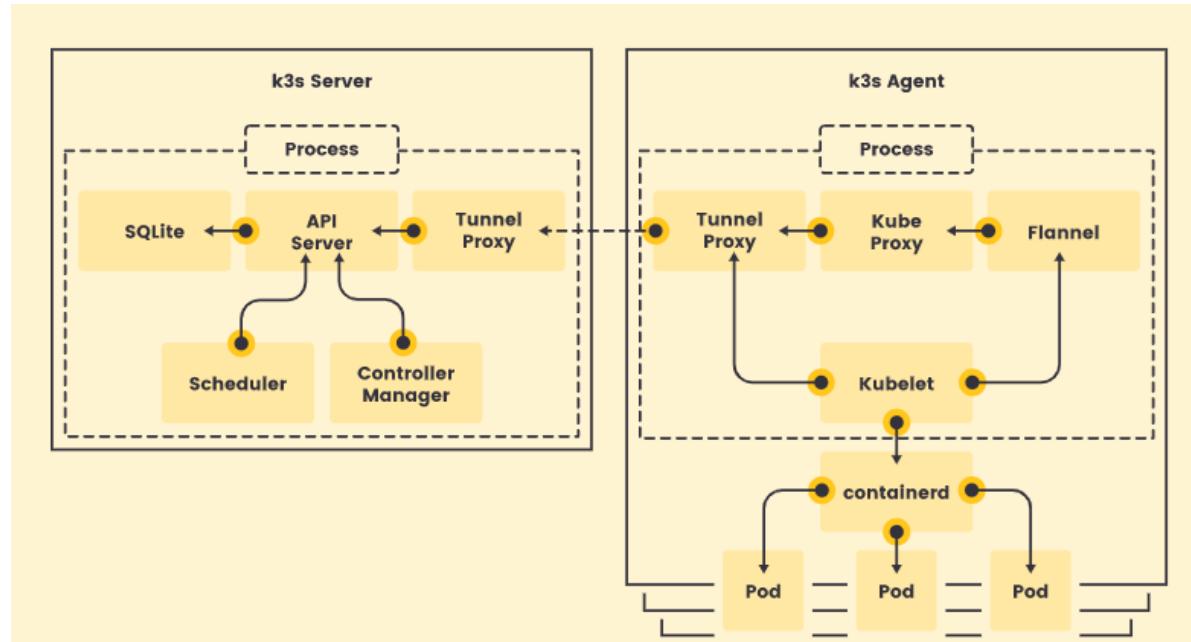
K3S

k3s Process Architecture

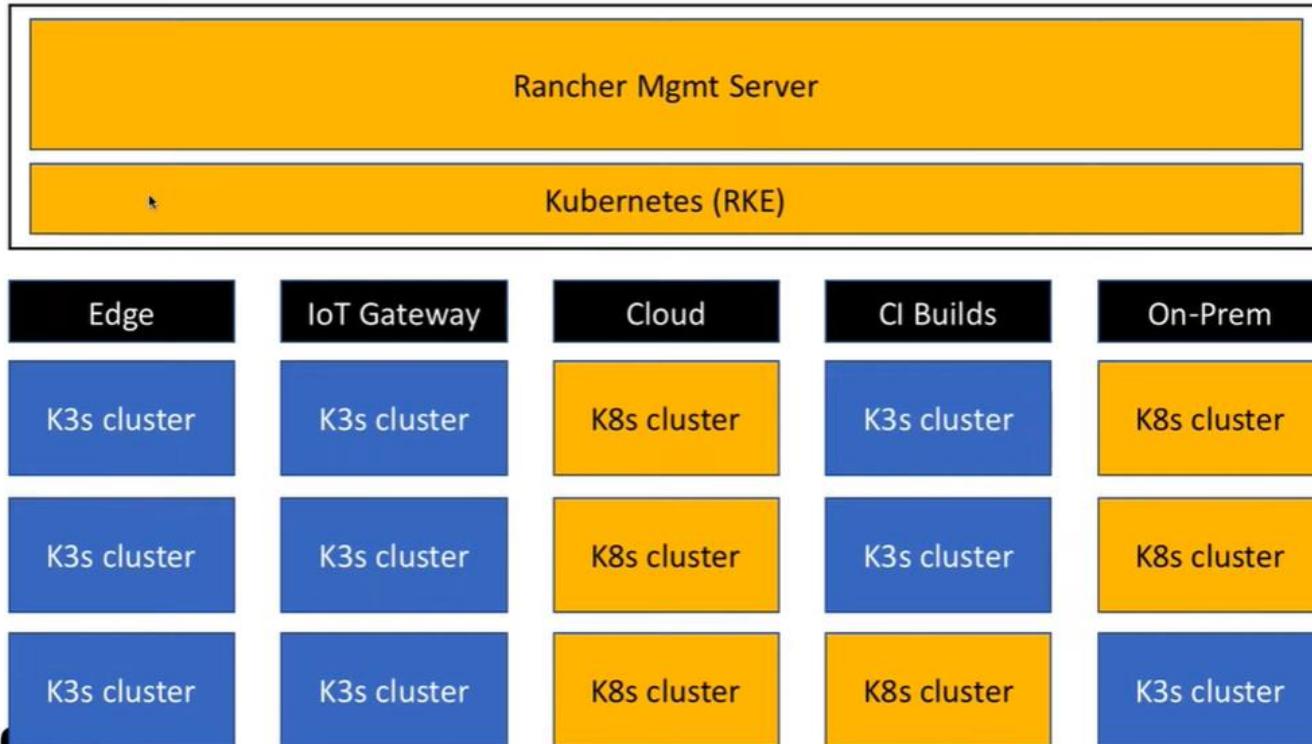
k3s is to get a very efficient and lightweight fully compliant Kubernetes distribution.

K3s can be install either through a simple script that will download and configure a linux binary

K3s.io



k3s Architecture



Why k3s?

Important distinctions

Lightweight Kubernetes

- Single Binary Side (-50mb)
- MemorySize (-300mb)
- Low Cognitive Load
- Perfect for Edge
- Used Just about everywhere

Designed for production

- Fully CNCF
- Secure by default
- Best practice defaults

Kubernetes Distribution

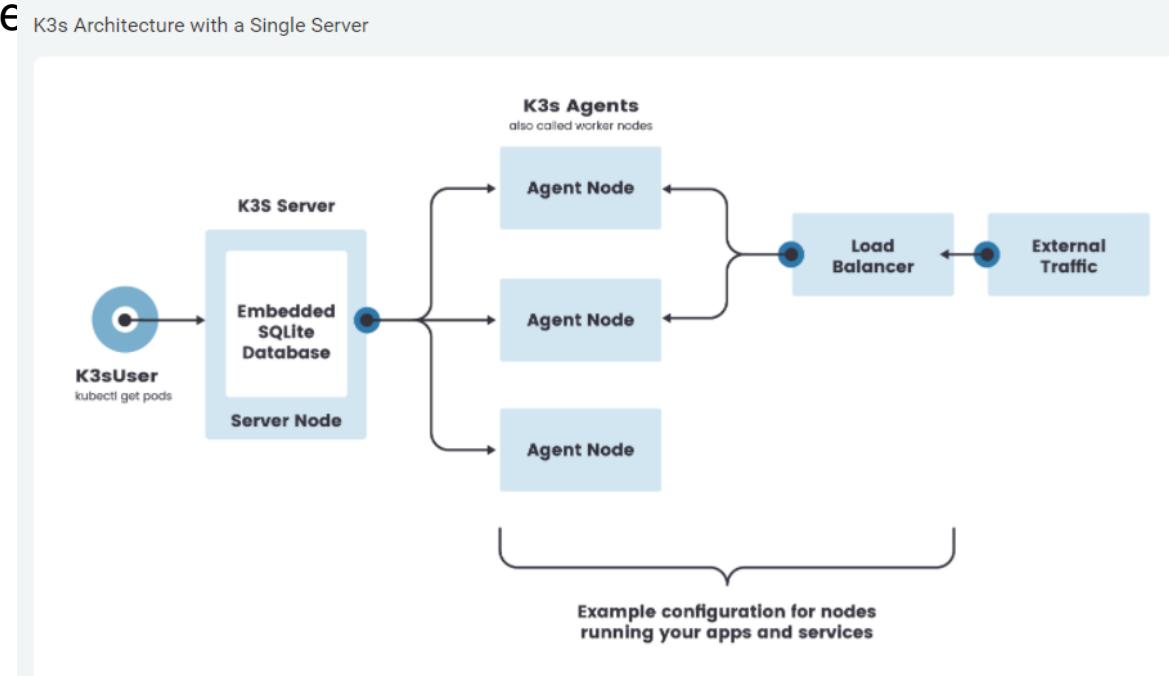
- Opinionated
- Complete
- Strong focus on simplicity

K3S Single Server

a cluster that has a single-node K3s server with an embedded SQLite database.

each agent node is registered to the same server node.

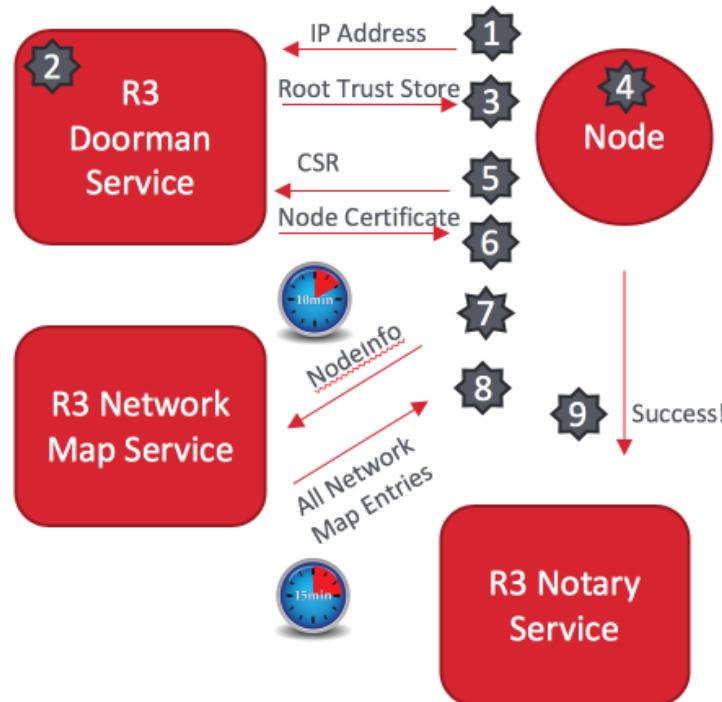
A K3s user can manipulate Kubernetes resources by calling the K3s API on the server node.



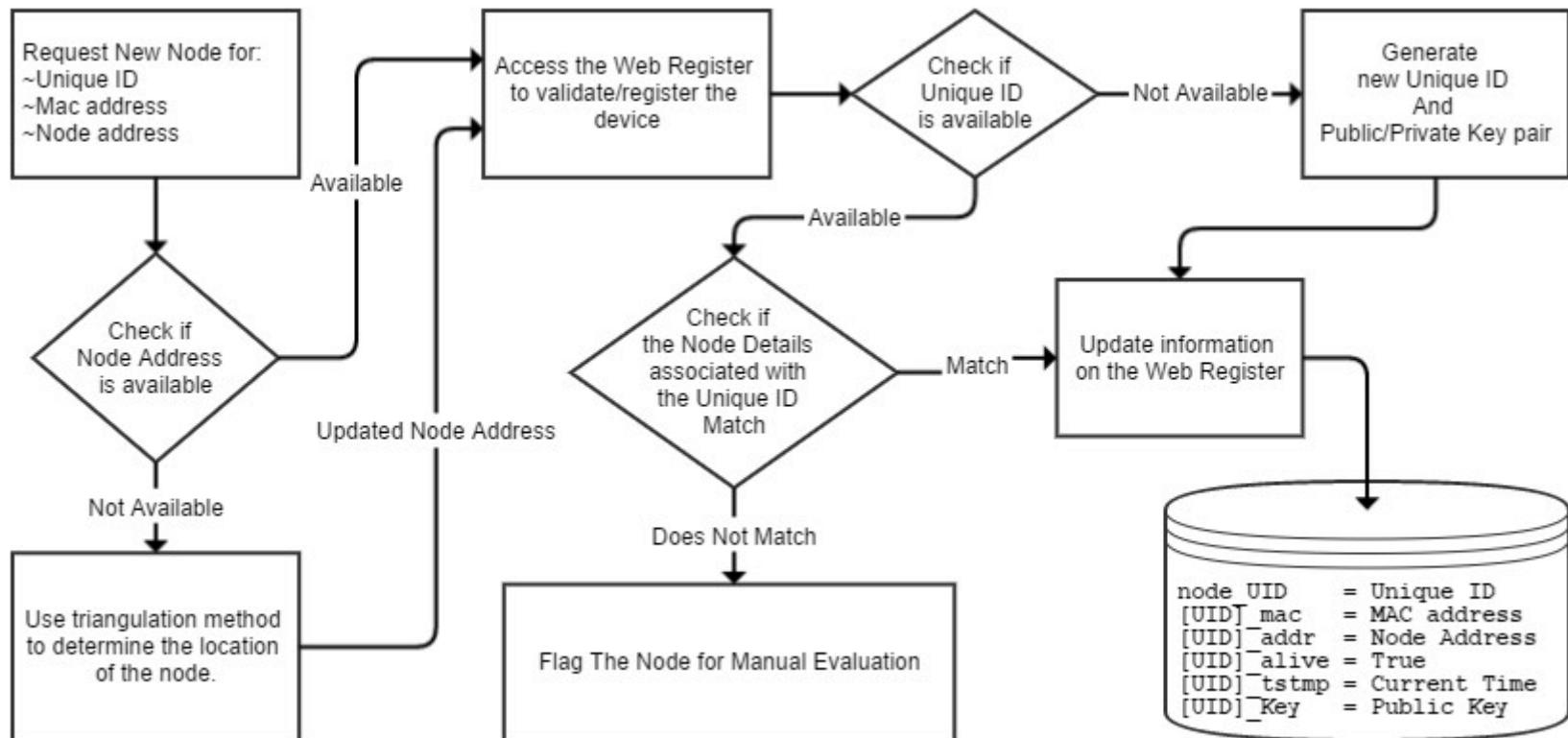
Node Registration

- Agent nodes are registered by the **k3s agent** process
- Connection is maintained by a client-side load balancer running as part of the agent process.
- Agents will register with the server
- Server stores node passwords
- Use the **--with-node-id** flag for node uniqueness

Node Registration



Node Registration



Where is k3s going?

The screenshot shows a GitHub pull request page for issue #447. The URL in the address bar is <https://github.com/cncf/toc/pull/447>. The page title is "Submit k3s for inclusion to CNCF as a sandbox project". The pull request has been merged by amye 22 days ago. The repository navigation bar includes links for Code, Issues (46), Pull requests (24, highlighted), Actions, Projects (2), Security, and Insights.

Submit k3s for inclusion to CNCF as a sandbox project
#447

Merged amye merged 1 commit into `cncf:master` from `cjellick:k3s-sandbox` 22 days ago

What is k3d?

K3d k3d is a binary that provisions a k3s kubernetes cluster on docker. Designed to easily run k3s in Docker, it provides a simple CLI to create, run, delete a full compliance Kubernetes cluster with 0 to n worker nodes.



K3d CNCF Landscape



Where can I find k3d?

<https://k3d.io/>

use the install script to grab the latest release:

- wget: wget -q -O -
<https://raw.githubusercontent.com/rancher/k3d/main/install.sh> | bash
- curl: curl -s
<https://raw.githubusercontent.com/rancher/k3d/main/install.sh> | bash



K3d

1. Runs k3s inside containers for development
2. Creates new cluster in seconds
3. Multiple clusters, multi node cluster
4. Runs on Docker (Docker for Desktop is the easiest route)

K3d Creating a Cluster

Create the directory on the host where we will persist the data:

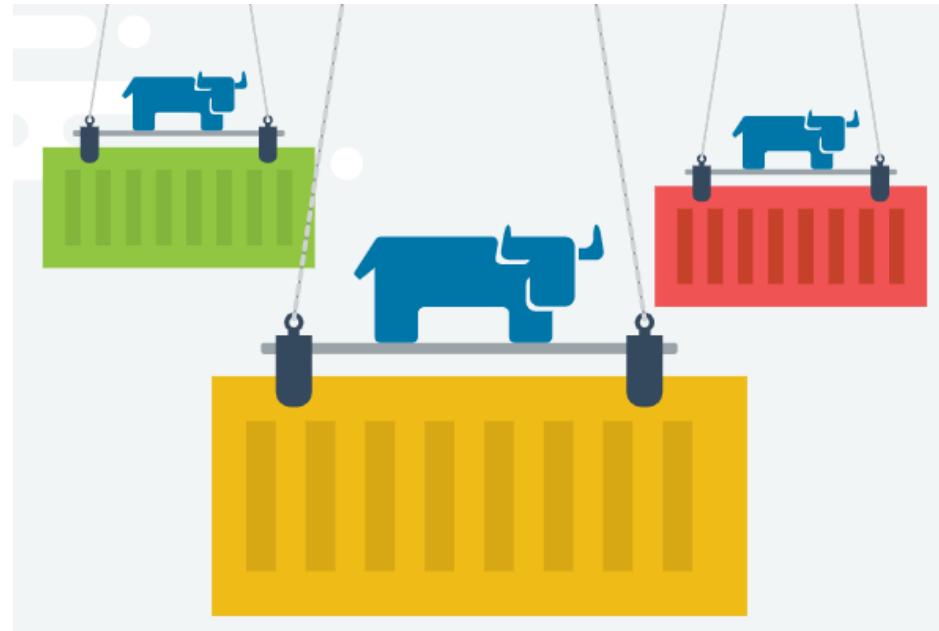
```
1 > mkdir -p /tmp/k3dvol
```

Create the cluster:

```
1 > k3d create --name "k3d-cluster" --volume /tmp/k3dvol:/tmp/k3dvol --publish "80:80" --workers :  
2 > export KUBECONFIG="$(k3d get-kubeconfig --name='k3d-cluster')"
```

K3d Code Home

<https://github.com/rancher/k3d/releases>



K3d Command Tree

<https://k3d.io/usage/commands/#command-tree>

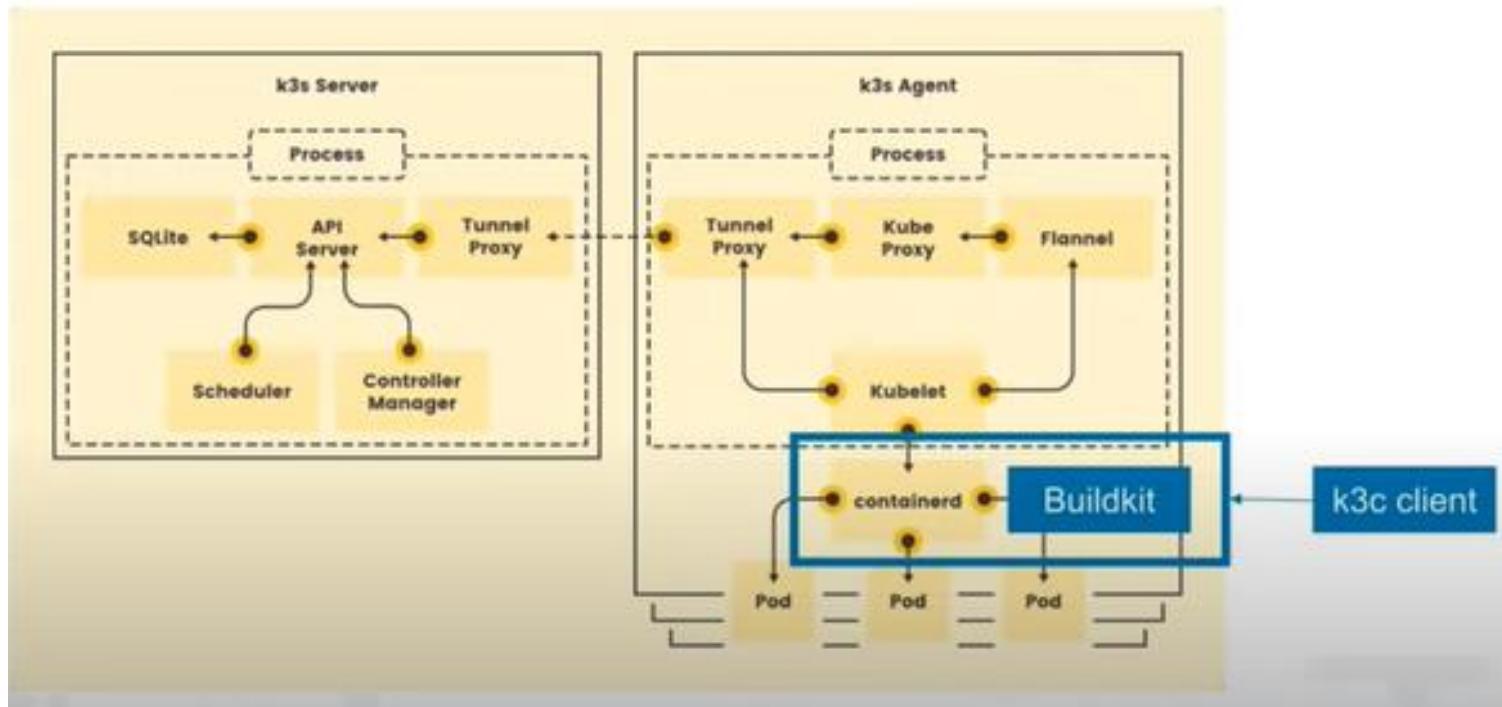


What is k3c

1. Enhances development and debugging flows
 - a. Familiar Docker CLI Style UX
 - b. Provide “build” functionality
 - c. Run one off containers for development “run”
 - d. Easy to debug nodes containers “ps, logs, exec”
2. New default container runtime for k3s
 - a. Conainerd + buildkit +enhanced CRI

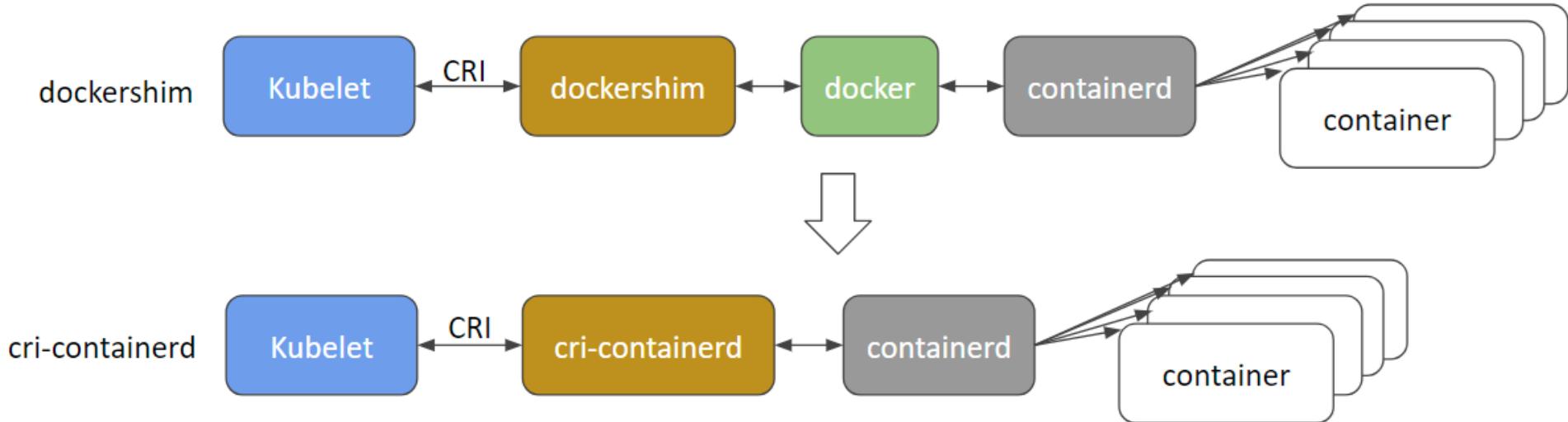


K3c in a context



Containerd

[Containerd](#) is an [OCI](#) compliant core container runtime designed to be embedded into larger systems. It provides the minimum set of functionality to execute containers and manages images on a node.



K3c and Buildkit

Docker Build introduced BuildKit, with improvements in performance, storage management, feature functionality, and security.

- Docker images created with BuildKit can be pushed to Docker Hub just like Docker images created with legacy build
- the Dockerfile format that works on legacy build will also work with BuildKit builds
- The new `--secret` command line option allows the user to pass secret information for building new images with a specified Dockerfile

K3c Architecture

k3c runs as a daemon either as root or one per user for rootless support.
NOTE: rootless isn't currently working, that's just the design right now. The daemon exposes a GRPC API. For building the buildkit API is just exposed directly from the k3c socket. containerd, buildkitd, and containerd-cri are all embedded directly into the k3c binary.



K3c Builds

Builds are currently run via k3c if it is available on the \$PATH otherwise docker is assumed to be available.

To build the k3c binary:

```
make build
```

To build the k3c image:

```
make package
```

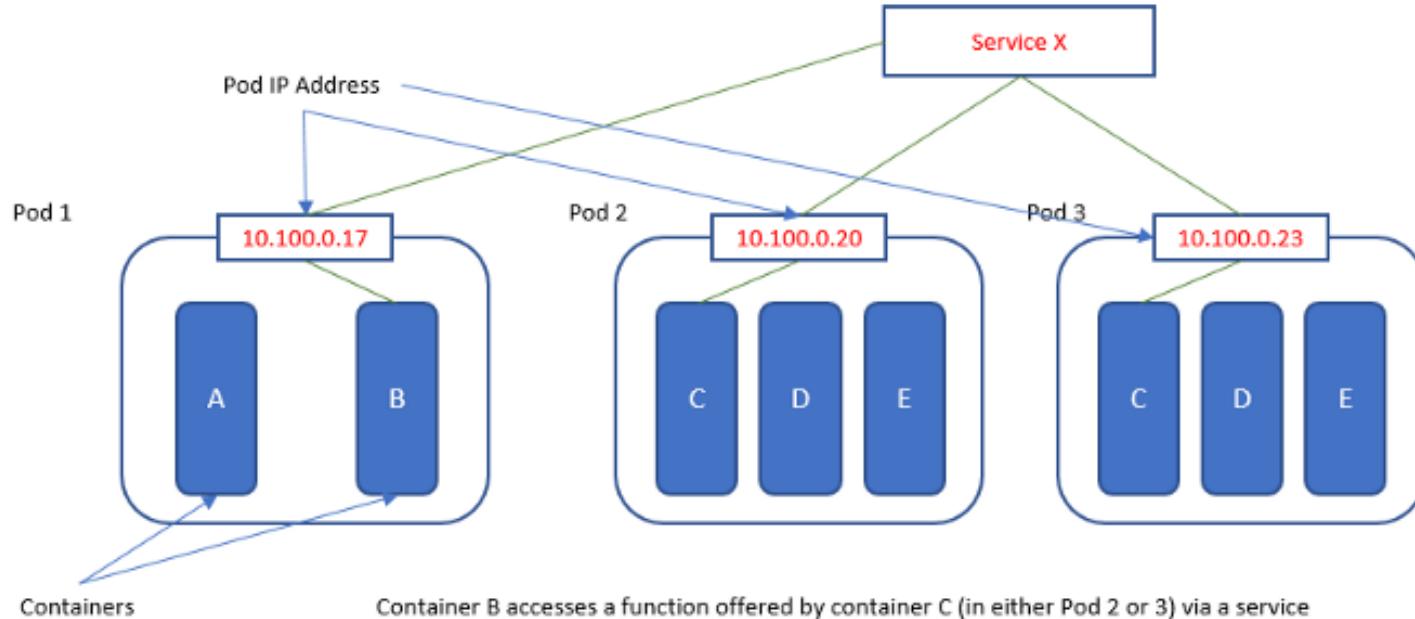


k3c + k3d

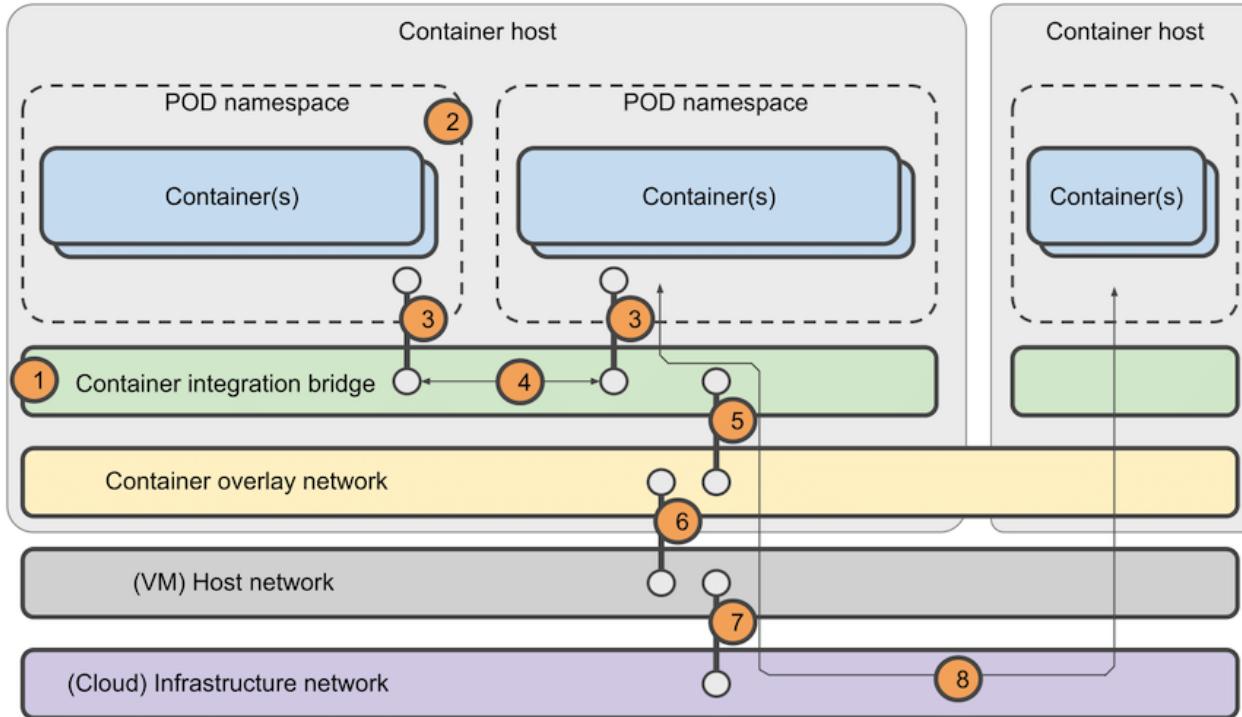
1. Coming Soon
2. “K3d build”
3. Run k3d on k3c itself



Debugging Side by Side with K8S

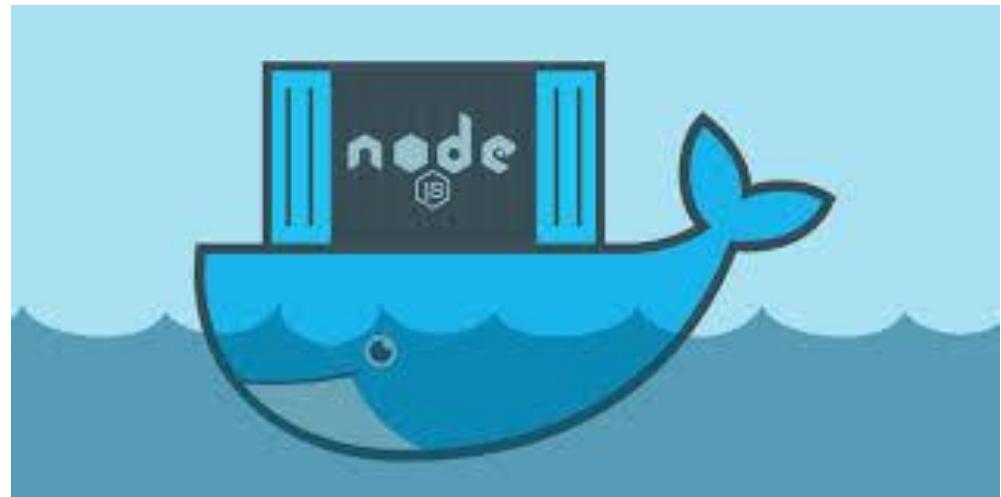


Networking with Kubernetes



Debugging Local Containers

Local Process with Kubernetes allows you to run and debug code on your development computer, while still connected to your Kubernetes cluster with the rest of your application or services.



Scalability

Our scalability definition is built on two concepts:

- Service Level Indicators
- Service Level Objectives

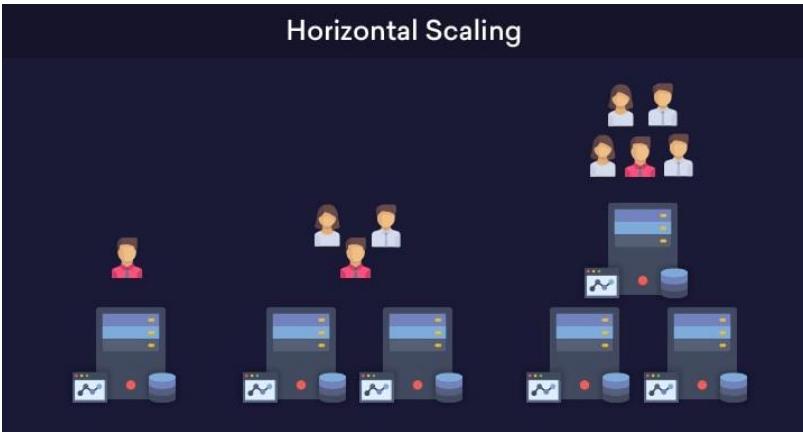


Vertically Scalable Components

vertical scaling (a.k.a. “**scaling up**”), you’re adding more power to your existing machine. In **horizontal scaling** (a.k.a. “**scaling out**”), you get the additional resources into your system by adding more machines to your network, sharing the processing and memory workload across multiple devices.



Horizontally Scalable Components



horizontally scalable refers to systems whose capacity and throughput are increased by adding additional nodes. This is in distinction to **vertically** scaled systems, where adding capacity and throughput generally involves replacing smaller nodes with larger and more powerful ones.

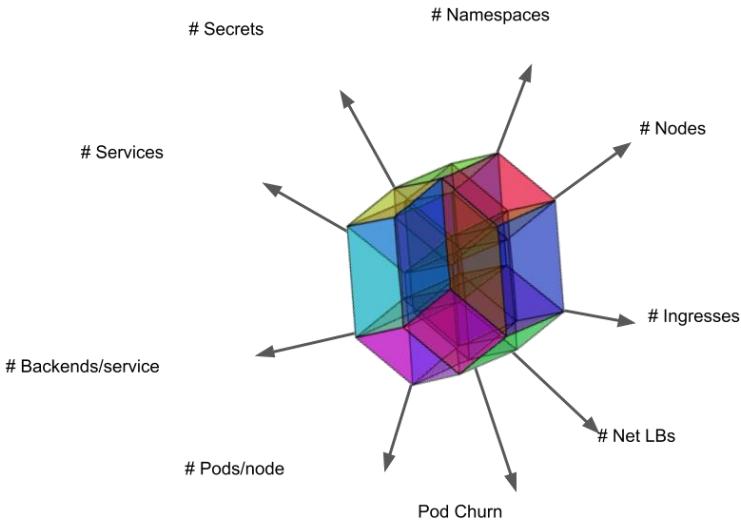
Building Applications Using Layers of Horizontally Scalable Components



Scalability Thresholds

To make the cluster eligible for SLO, users also can't have too many objects in their clusters. More concretely, the number of different objects in the cluster **MUST** satisfy thresholds defined in [thresholds file](#)

One of the prerequisites for SLOs being satisfied is keeping the load in the cluster within recommended limits.



Source of hypercube image: <http://www.gregegan.net/APPLETS/29/29.html>

Container Network Spaghetti

1. Highly-coupled container-to-container communications => linux
2. Pod-to-Pod communications => various
3. Pod-to-Service communications => service
4. External-to-Service communications => service or ingress



POP QUIZ:

Kubernetes: Local Development, Test, and Debug



What is horizontal scalability?

- A: adding capacity and throughput generally involves replacing smaller nodes
- B: Systems whose capacity and throughput are increased by adding additional nodes.
- C: the capacity to be changed in size or scale.

POP QUIZ:



Kubernetes: Local Development, Test, and Debug



What is horizontal scalability?

A: adding capacity and throughput by replacing smaller nodes

B: Systems whose capacity and throughput are increased by adding additional nodes.

C: the capacity to be changed in size or scale.

POP QUIZ:

Kubernetes: Local Development, Test, and Debug



What is K3c?

- A: A binary that provisions a kubernetes cluster on docker.
Designed to easily run in Docker
- B: Lightweight Kubernetes Distribution
- C: A local container engine designed to fill the same gap Docker does in the Kubernetes ecosystem.

POP QUIZ:

Kubernetes: Local Development, Test, and Debug



What is K3c?

A: A binary that provisions a kubernetes cluster on docker.
Designed to easily run in Docker

B: Lightweight Kubernetes Distribution

C: A local container engine designed to fill the same gap Docker does in the Kubernetes ecosystem.

Experiment – k3d Getting Started



Experiment – k3d and PVC

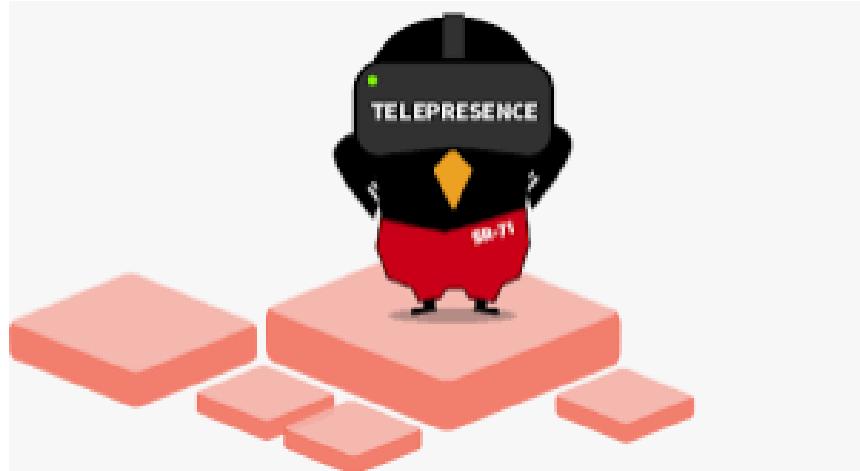


Remote Development, Test, and Debug



Telepresence

telepresence is an open source tool that lets you run a single service locally, while connecting that service to a remote Kubernetes cluster.

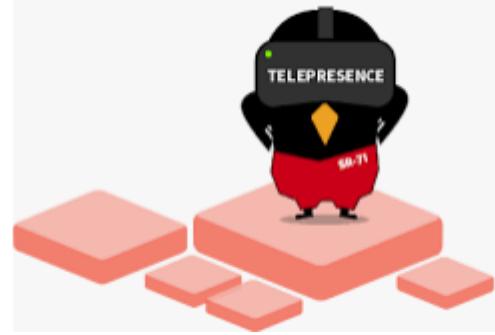


Using telepresence to run local code in a remote server

You can run a local process using Telepresence that can access that service, even though the process is local but the service is running in the Kubernetes cluster:

```
$ telepresence --run curl http://myservice:8000/
```

Hello, world!



Attaching a debugger to a running container on a remote server

This debugging session type allows you to connect the IDE to a remote target and debug a process running there. By remote target, we can consider containers running on the local machine remote targets or actual servers either on-premise or in the cloud.

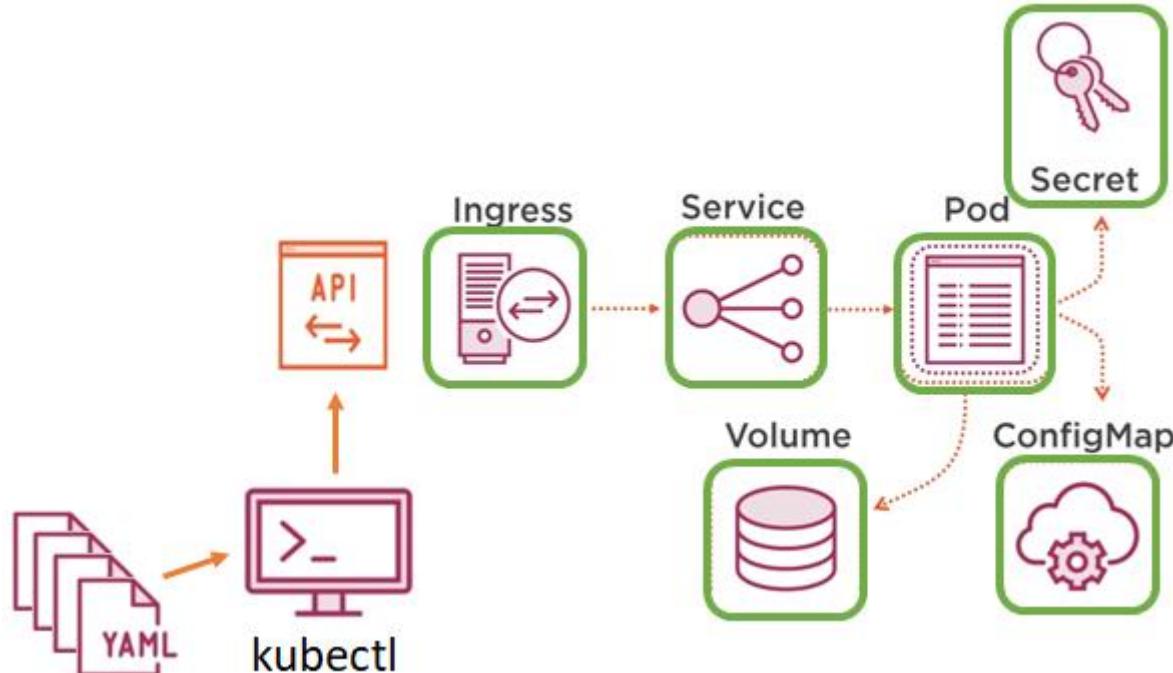


Attaching a debugger to a running container on a Local machine

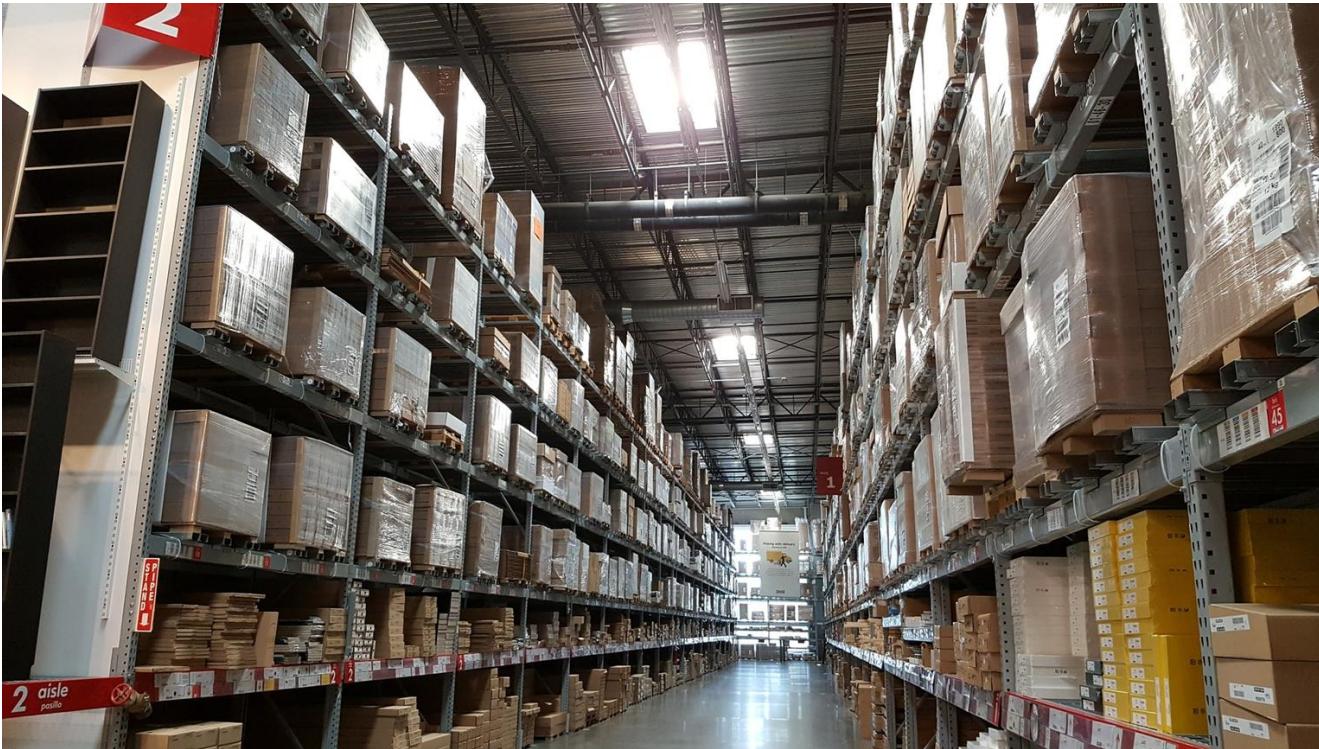
To ensure that the debugging session is successful and you can debug the application without issues, all you need to do is to compile your application with a special flag. The IDE will add these flags automatically for the other configuration types, so these are necessary only when compiling the application manually.



Kubernetes Does Everything



Problem Statement



Does Kubernetes need a package manager?

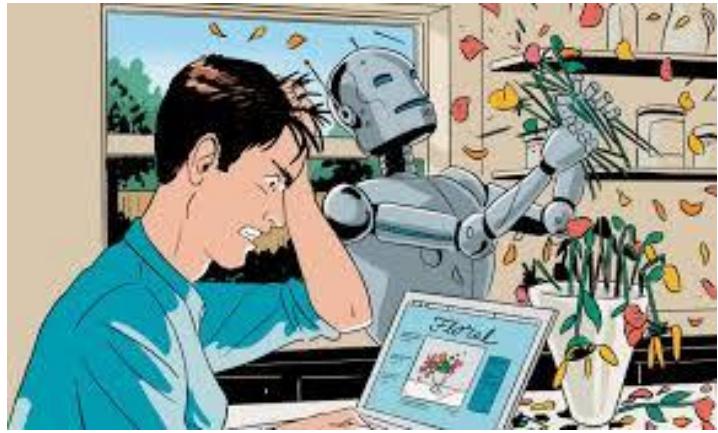


YUM

Humans on the other hand ...

tend to struggle with YAML manifests which can be hundreds or thousands of lines long. Optimally we don't want to maintain manifests for every permutation of an application or manage its versioning and lifecycle.

We'd much rather offload those tasks to a reliable (and automatable) tool with the flexibility to modify parameters for us when instructed to do so.



What is Helm

Helm with Kubernetes : Helm helps in combining several Kubernetes factors such as service, deployments, configmaps; many more to the single unit known as Helm Charts. The Cloud Native Computing Foundation maintains the latest version of Helm in association with Microsoft and Google.



Installing Helm

Install Helm on Windows using the following command:

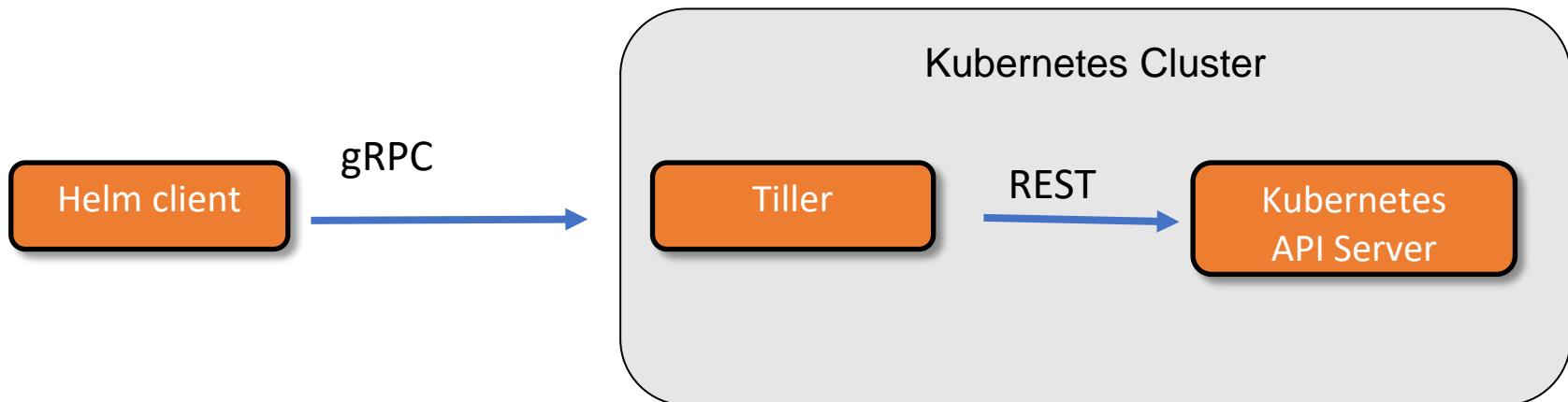
```
> choco install kubernetes-helm
```

Install Helm on macOS using the following command:

```
$ brew install helm
```

Helm Architecture

- Helm client
 - command-line
 - Interacts with Tiller
 - Local chart development
- Tiller
 - In-cluster
 - Listens to Helm client
 - Interacts with Kube API
 - Manages lifecycle



Helm Features



Charts



Templates



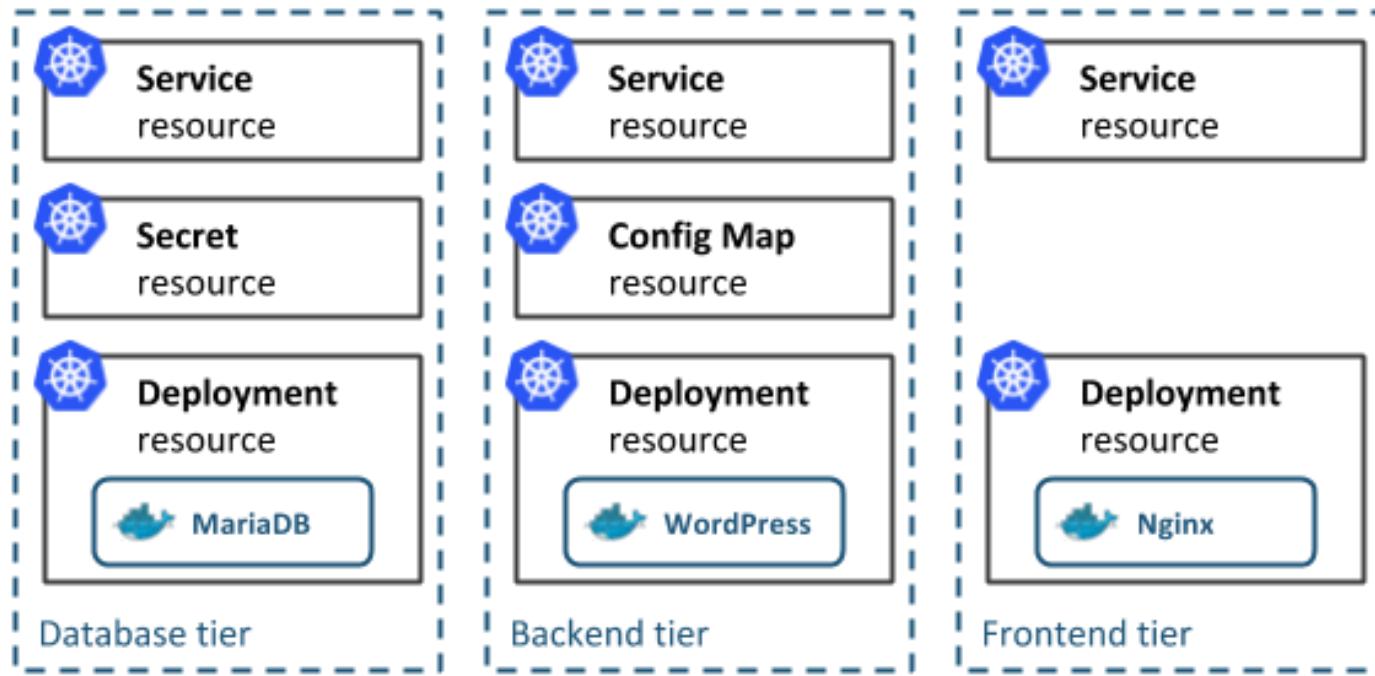
Dependencies



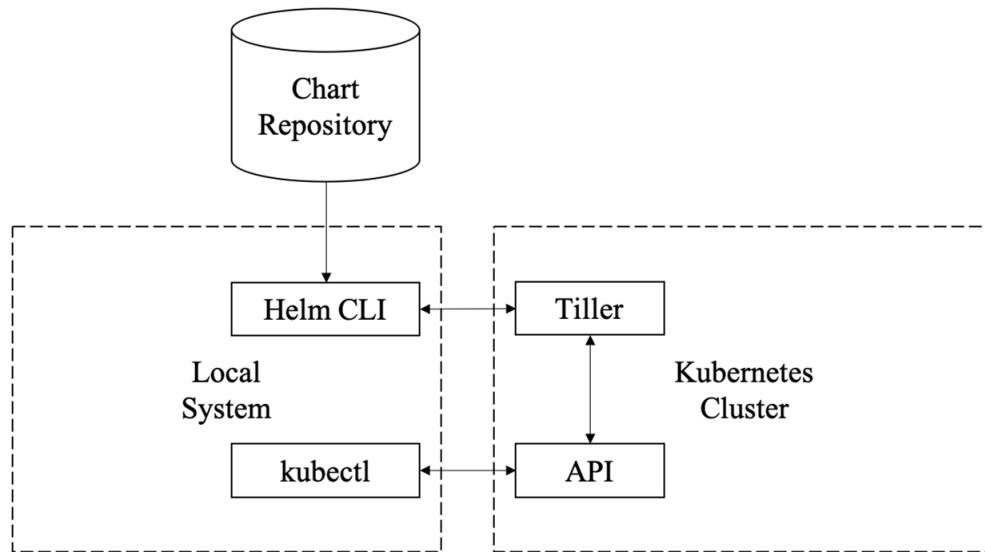
Repositories

Helm Chart

Helm uses a packaging format called **charts**. A **chart** is a collection of files that describe a related set of Kubernetes resources

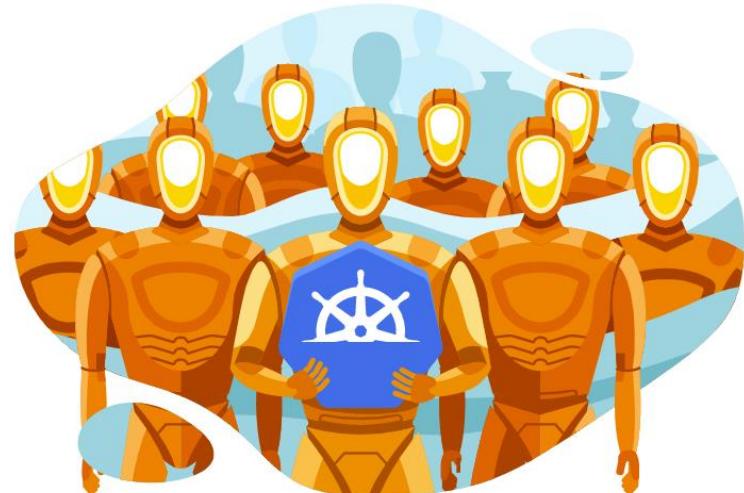


Helm Charts



Helm Chart Structure

- nginx-demo
 - Chart.yaml
 - README.md
 - requirements.yaml
- templates
 - deployment.yaml
 - ingress.yaml
 - service.yaml
 - NOTES.txt
 - tests
 - test-connection.yaml
- values.yaml
- Define sub-charts and dependencies



Helm Chart Structure

Under the top-level directory are the files and directories that comprise the Helm chart. The following table shows each of the possible files and directories.

File/directory	Definition	Required?
Chart.yaml	A file that contains metadata about the Helm chart.	Yes.
templates/	A directory that contains Kubernetes resources in YAML format.	Yes, unless dependencies are declared in Chart.yaml.
templates/NOTES.txt	A file that can be generated to provide usage instructions during chart installation.	No.
values.yaml	A file that contains the chart's default values.	No, but every chart should contain this file as a best practice.
.helmignore	A file that contains a list of files and directories that should be omitted from the Helm chart's packaging.	No.

Helm Chart.yaml

Chart.yaml

```
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: nginx-demo
version: 0.1.0
```

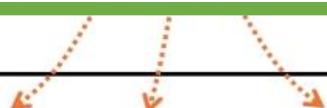
Major, Minor, Patch
(SemVer 2.0)

- App you are installing

Helm Chart.yaml

Chart.yaml

```
apiVersion: v1
appVersion: "1.0"
description: A Helm chart for Kubernetes
name: nginx-demo
version: 0.1.0
```



Major, Minor, Patch
(SemVer 2.0)

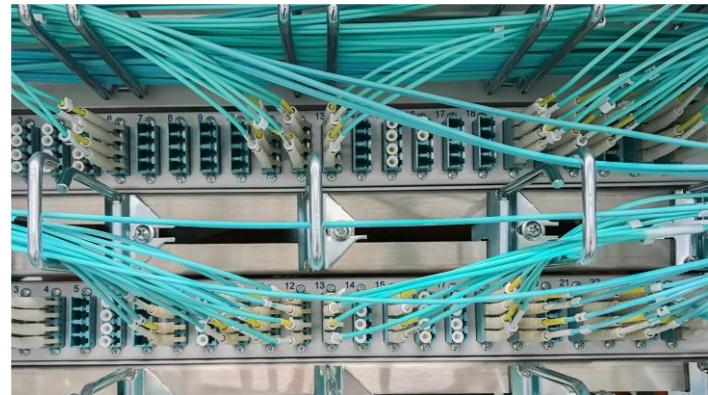
- Version of Helm chart

Helm Commands

Action	Command
Install a Release	<code>helm install</code>
Upgrade a Release revision	<code>helm upgrade [release]</code>
Rollback to a Release revision	<code>helm rollback [release]</code>
Print Release history	<code>helm history [release]</code>
Display Release status	<code>helm status [release]</code>
Show details of a release	<code>helm get [release]</code>
Uninstall a Release	<code>helm delete</code>
List Releases	<code>helm list</code>

Understanding WordPress Application

1. Highly-coupled container-to-container communications => linux
2. Pod-to-Pod communications => various
3. Pod-to-Service communications => service
4. External-to-Service communications => service or ingress



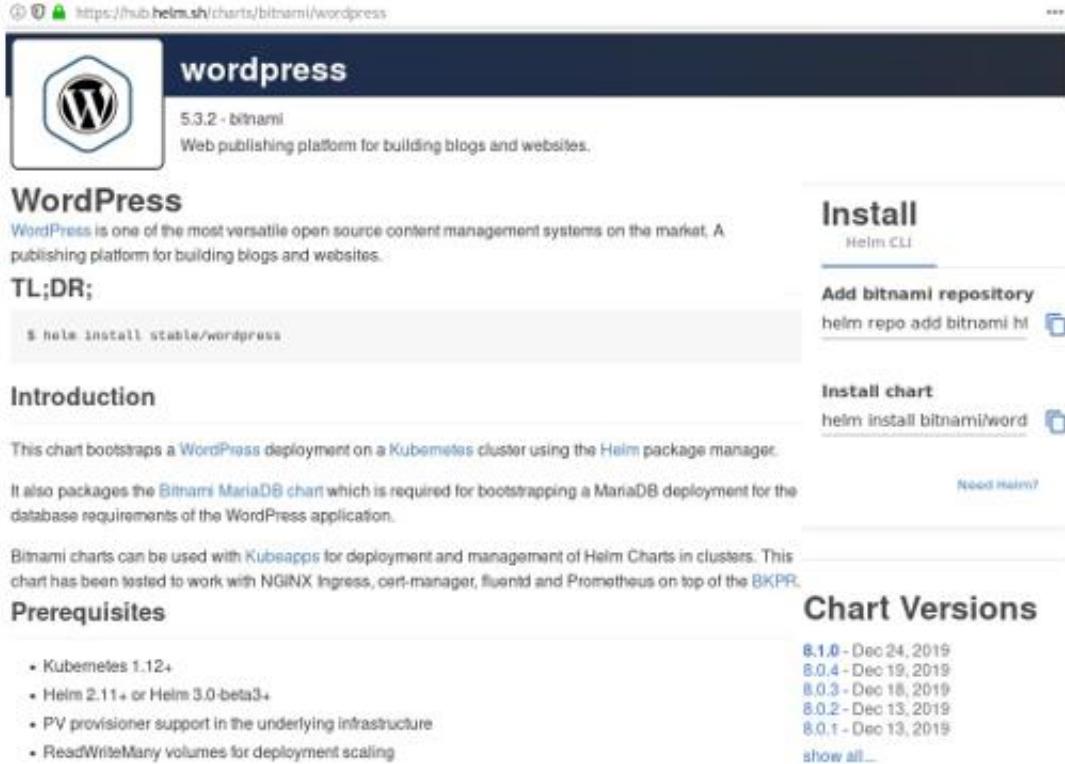
Finding a WordPress Chart

1. Let's search Helm Hub for any existing WordPress charts. Each chart in Helm Hub has a set of keywords that can be searched against. Execute the following command to locate charts containing the `wordpress` keyword:

Upon running this command, an output similar to the following should be displayed:

URL	CHART VERSION	APP VERSION	DESCRIPTION
https://hub.helm.sh/charts/bitnami/wordpress	8.1.0	5.3.2	Web publishing
https://hub.helm.sh/charts/presslabs/wordpress-...	v0.6.3	v0.6.3	Presslabs Word
https://hub.helm.sh/charts/presslabs/wordpress-...	v0.7.4	v0.7.4	A Helm chart

Viewing WordPress Chart



The screenshot shows the Helm Hub page for the WordPress chart. At the top, there's a navigation bar with icons for search, refresh, and user profile, followed by the URL <https://hub.helm.sh/charts/bitnami/wordpress>. Below the header, the chart title "wordpress" is displayed in a large font, with "5.3.2 · bitnami" underneath. A brief description follows: "Web publishing platform for building blogs and websites." On the left side, there's a "WordPress" section with a brief description and a "TL;DR;" section containing the command `$ helm install stable/wordpress`. The main content area has a heading "Introduction" with text about bootstrapping a WordPress deployment on a Kubernetes cluster using Helm. It also mentions packaging the Bitnami MariaDB chart for MariaDB deployment. Below this, there's a "Prerequisites" section listing requirements: Kubernetes 1.12+, Helm 2.11+ or Helm 3.0-beta3+, PV provisioner support, and ReadWriteMany volumes for deployment scaling. On the right side, there's an "Install" section with a "Helm CLI" link and a "Add bitnami repository" button. Below that is an "Install chart" section with a "helm install bitnami/word" button. At the bottom, there's a "Chart Versions" section showing four versions: 8.1.0 - Dec 24, 2019, 8.0.4 - Dec 19, 2019, 8.0.3 - Dec 18, 2019, and 8.0.2 - Dec 13, 2019, with a "show all..." link.

The WordPress chart's page from Helm Hub provides many details, including the maintainer of the chart

Helm Dependencies

This Chart.yaml example declares two dependencies

```
# Chart.yaml
dependencies:
- name: nginx
  version: "1.2.3"
  repository: "https://example.com/charts"
- name: memcached
  version: "3.2.1"
  repository: "https://another.example.com/charts"
```

The 'name' should be the name of a chart, where that name must match the name in that chart's 'Chart.yaml' file.

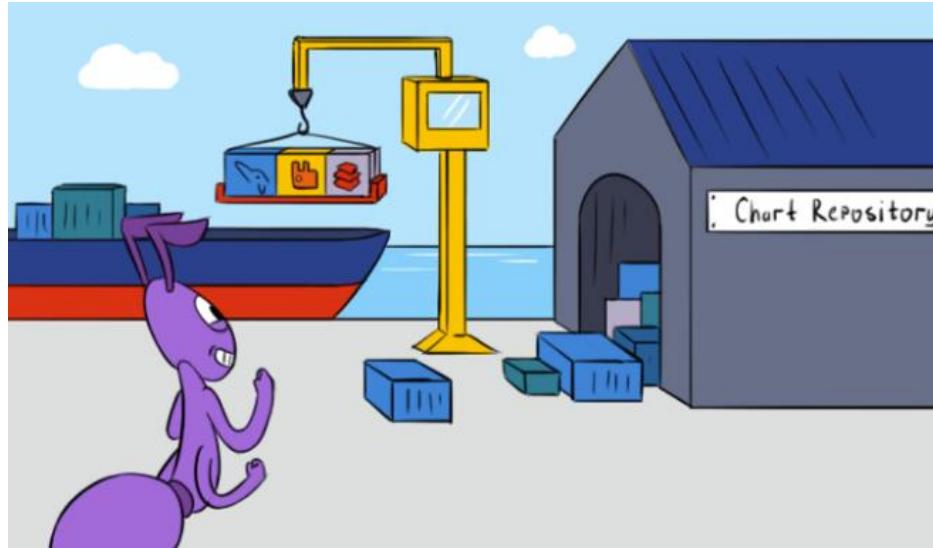
The 'version' field should contain a semantic version or version range.

Helm Repositories

For example, the layout of the repository

<https://example.com/charts> might look like this:

```
charts/
  |
  |- index.yaml
  |
  |- alpine-0.1.2.tgz
  |
  |- alpine-0.1.2.tgz.prov
```



Helm Templates

Helm includes many template functions you can take advantage of in templates.

They are listed here and broken down by the following categories:

- Cryptographic and Security
- Date
- Dictionaries
- Encoding
- File Path
- UUID
- Kubernetes and Chart Control
- Logic and Flow Control
- Lists
- Math
- Network
- Reflection
- Regular Expressions
- Semantic Versions
- String
- Type Conversion
- URL

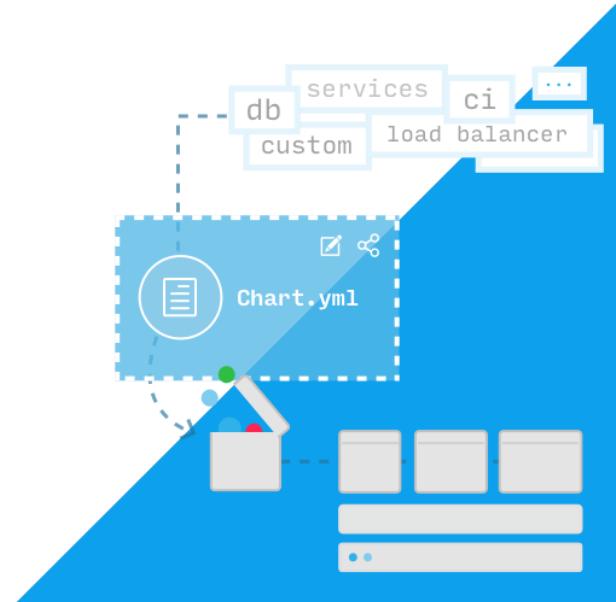
Linting Helm Charts and Templates

Linting your charts is important to prevent errors in your chart's formatting or the chart's definition file and provide guidance on best practices when working with Helm charts.

The helm lint command has the following syntax:

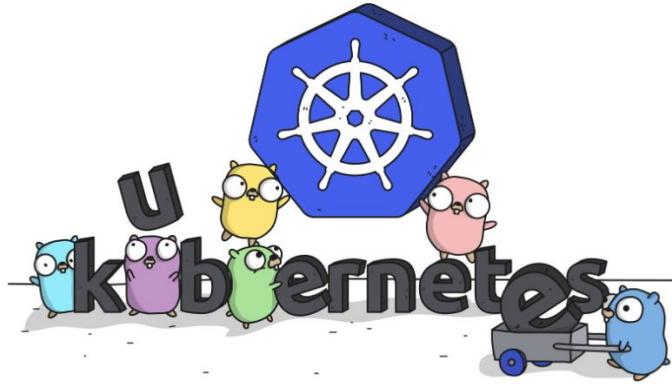
```
$ helm lint PATH [flags]
```

The helm lint command is designed to be run against a chart directory to ensure that the chart is valid and properly formatted.

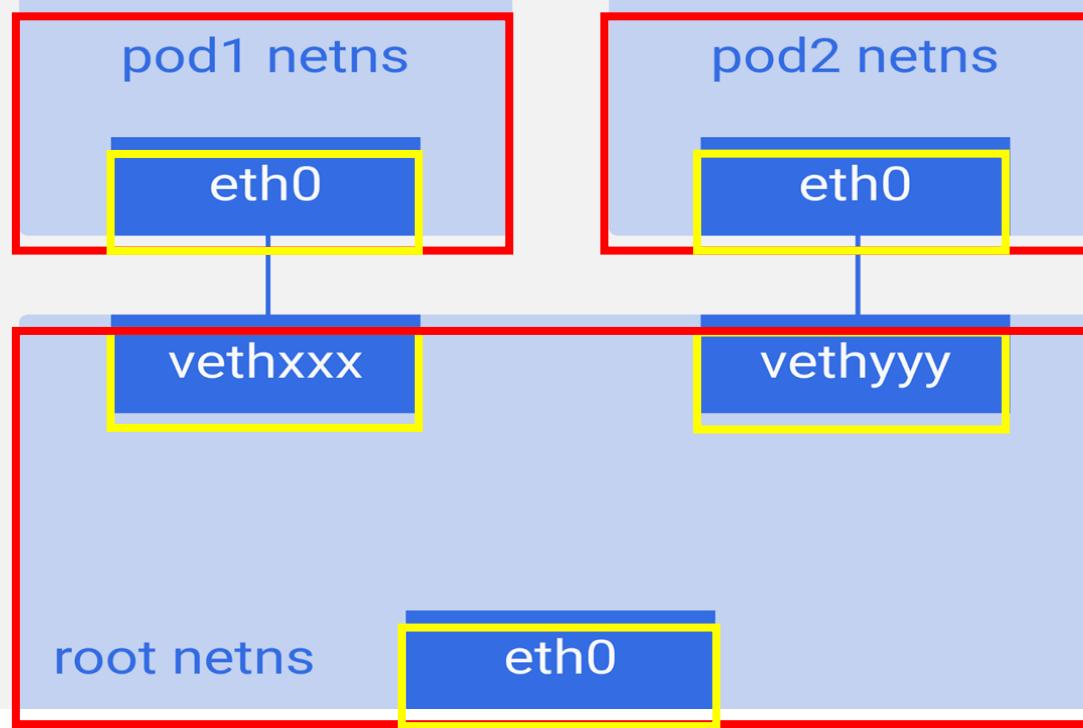


Debugging Templates

Debugging templates can be tricky because the rendered templates are sent to the Kubernetes API server, which may reject the YAML files for reasons other than formatting.



Node Namespaces



Experiment – k3d, Helm, Rancher



Package Management

Options & Kept



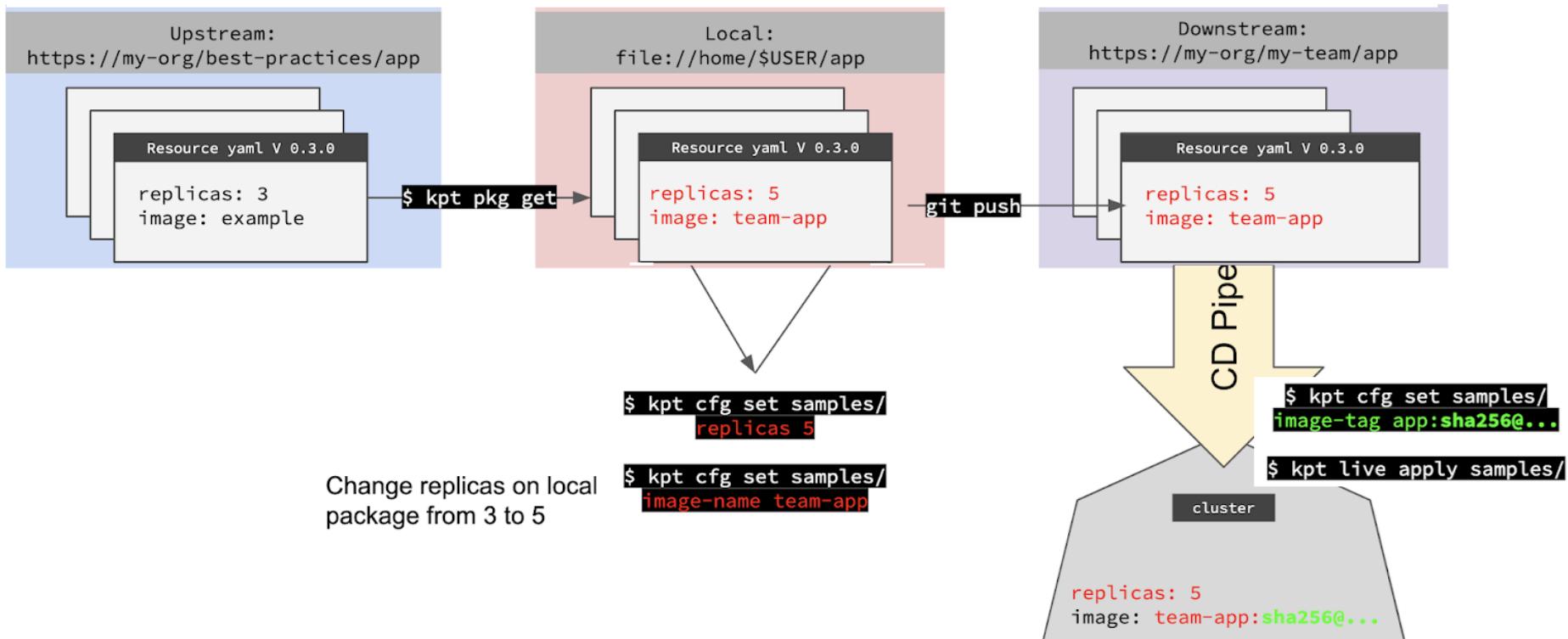
Packaging Wheel of Fortune



What is KPT

- ❖ An OSS tool for Kubernetes packaging, which uses a standard format to bundle, publish, customize, update, and apply configuration manifests.
- ❖ Released into the wild about the same time as the Pandemic

What is KPT



KPT Installation

- Gcloud
- Homebrew
- Source Code
- Docker
- Binaries

<https://googlecontainertools.github.io/kpt/installation/binaries/>

KPT Architecture

Kpt is built around an “as data” architecture bundling [Kubernetes resource configuration](#), a format for both humans and machines. The ability for tools to read and write the package contents using standardized data structures enables powerful new capabilities



KPT API Conventions

- Commands that read resources should be able to read them from files, directories or stdin
 - It should be possible to pipe `kubectl get -o yaml` to the input of these commands
- Commands that write resources should be able to write them to files, directories or stdout
 - It should be possible to pipe the output of these commands to `kubectl apply -f`
 -



KPT Setters

Creating a Setter

Setters may be created either manually (by editing the Kptfile directly), or programmatically (through the `create-setter` command). The `create-setter` command will:

1. create a new OpenAPI definition for a setter in the Kptfile
2. create references to the setter definition on the resource fields



KPT Packaging/ pkg

The two primary sets of capabilities that are required to enable reuse are:

1. The ability to distribute/publish/share, compose, and update groups of configuration artifacts, commonly known as packages.
2. The ability to adapt them to your use cases, which we call customization.



KPT cfg

```
# print the package using tree based structure
$ kpt cfg tree helloworld --name --image --replicas
helloworld
└── [deploy.yaml] Deployment helloworld-gke
    ├── spec.replicas: 5
    └── spec.template.spec.containers
        └── 0
            ├── name: helloworld-gke
            └── image: gcr.io/kpt-dev/helloworld-gke:0.1.0
└── [service.yaml] Service helloworld-gke
```



```
# List available setters
$ kpt cfg list-setters helloworld replicas
  NAME      DESCRIPTION      VALUE      TYPE      COUNT      SETBY
  replicas  'helloworld replicas'  5      integer  1
```

```
# set a high-level knob
$ kpt cfg set helloworld replicas 3
set 1 fields
```

KPT live

Live contains the next-generation versions of apply related commands for deploying local configuration packages to a cluster.



KPT FN

Examples

```
# run the function defined by gcr.io/example.com/my-fn as a local container
# against the configuration in DIR
kpt fn run DIR/ --image gcr.io/example.com/my-fn
```

```
# run the functions declared in files under FUNCTIONS_DIR/
kpt fn run DIR/ --fn-path FUNCTIONS_DIR/
```

```
# run the functions declared in files under DIR/
kpt fn run DIR/
```

KPT Functions

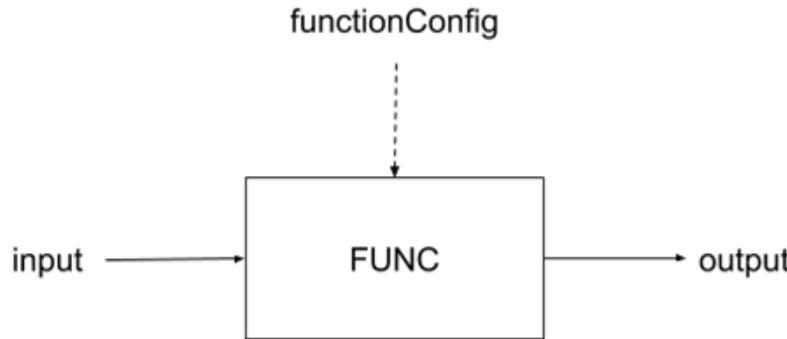
Config Functions are client-side programs that make it easy to operate on a repository of Kubernetes resource files.

Use cases:

Configuration Validation

Configuration Generation

Configuration Transformation

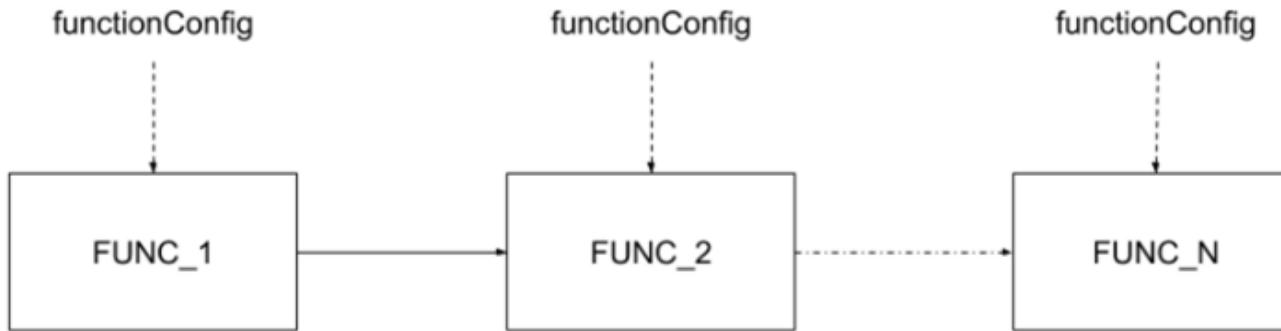


- **FUND:** A program that performs CRUD (Create, Read, Update, Delete) operations on the input. This program can be packaged as a container executable or starlark script.
input: A Kubernetes List type containing objects to operate on.
- Output: A Kubernetes List type containing the resultant Kubernetes objects.
- **functionConfig:** An optional Kubernetes object used to parameterize the function's behavior.

KPT Functions

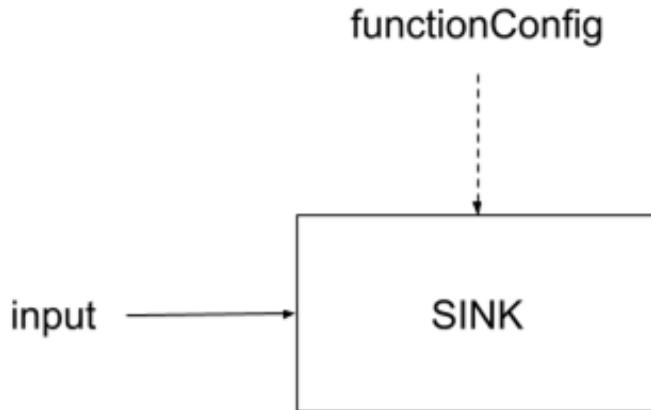
Pipeline

In order to do something useful with a function, we need to compose a pipeline with a source and a sink function

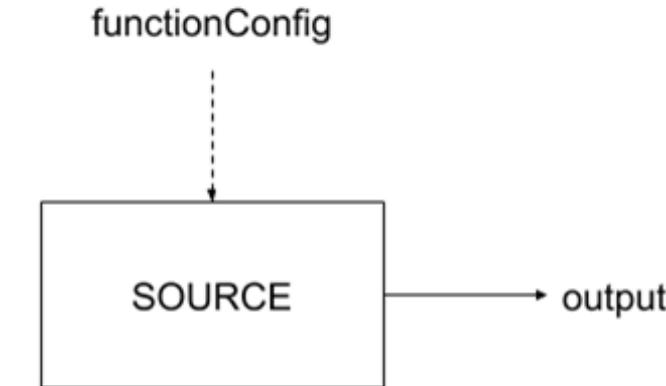


You can also use a container-based workflow orchestrator by exporting a workflow pipeline. Get detailed tutorials on how to use [`kpt fn export`] from the `export a workflow` guide.

KPT Functions



A Sink Function produces no output:
Instead, the function typically writes
configurations to an external system (e.g.
writing files to a filesystem)



A Source Function produces no input:
Instead, the function typically produces the output by
reading configurations from an external system (e.g.
reading files from a filesystem) writers configurations to
an external system (e.g. writing files to a filesystem)

Using helm and kpt

Helm charts may be used to generate kpt packages which can then be further customized directly.

Steps

1. Fetch a Helm chart
2. Expand the Helm chart
3. Publish the kpt package



Kpt installation sources

<https://googlecontainertools.github.io/kpt/installation/binaries/>



POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



What is helm?

- A: Helm the application package manager that runs on Kubernetes
- B: Helm is used for KPT Installation
- C: Helm is an open sourced system for running kubernetes



POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



What is helm?

A: Helm the application
package manager that runs on
Kubernetes

B: Helm is used for KPT
Installation

C: Helm is an open sourced
system for running kubernetes

POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



Why do you need Helm?

- A: To help you manage, examine, manipulate, customize, validate, and apply Kubernetes resource configuration files
- B: for running local Kubernetes clusters using Docker container “nodes”
- C: the ability to leverage Kubernetes packages through the click of a button or single CLI command

POP QUIZ:



Kubernetes: Remote Development, Test, and Debug



Why do you need Helm?

A: To help you manage, examine, manipulate, customize, validate, and apply Kubernetes resource configuration files

B: for running local Kubernetes clusters using Docker container “nodes”

C: the ability to leverage Kubernetes packages through the click of a button or single CLI command

POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



What does KPT stand for?

- A: Kubernetes package trainer
- B: KIND Path tutorial
- C: It is not an acronym



POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



What does KPT stand for?

- A: Kubernetes package trainer
- B: KIND Path tutorial
- C: It is not an acronym



POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



True or False.

Helm and KPT are two different ways to complete the same function.



POP QUIZ:

Kubernetes: Remote Development, Test, and Debug



True or **False.**

Helm and KPT are two different ways to complete the same function.



Experiment – Getting Started KPT



Experiment – KPT & Helm



Controller/ Operator Patterns

management, operations, automation



What is a Controller?

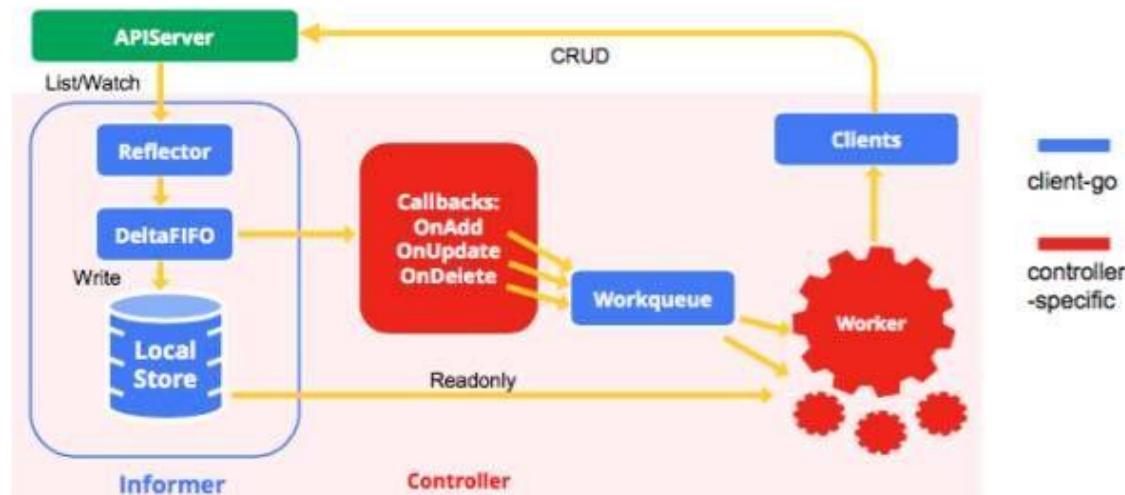


Controllers take care of routine tasks to ensure the desired state matches the observed state.

Controller pattern

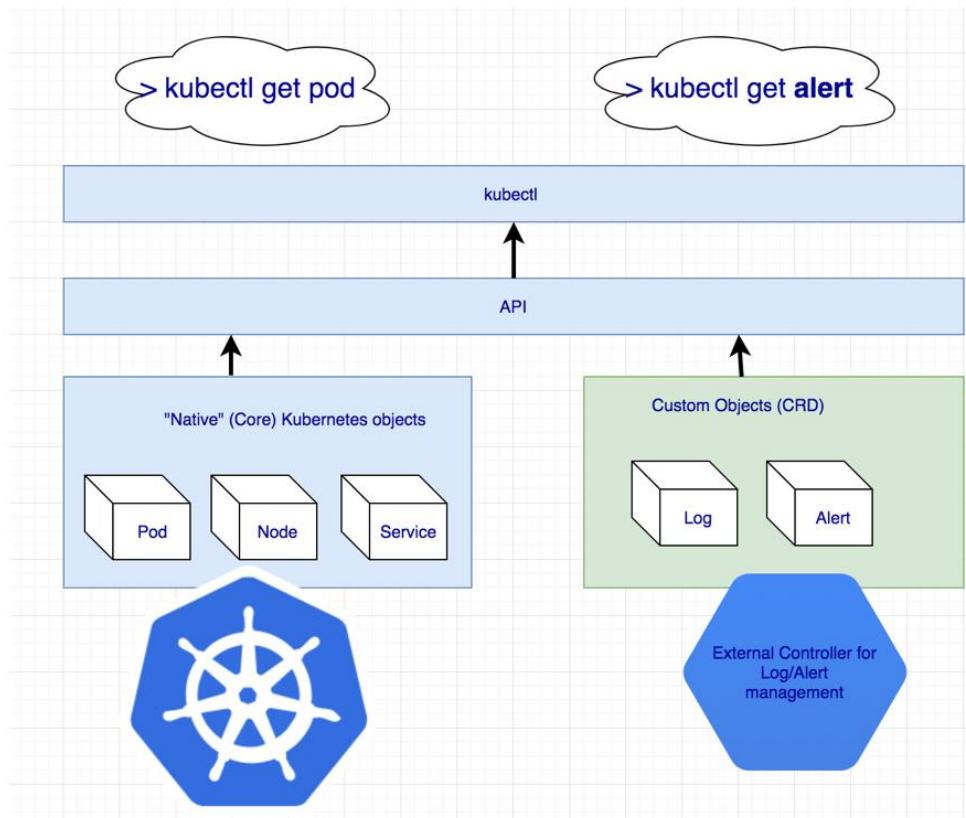
Kubernetes runs a group of controllers that take care of routine tasks to ensure the desired state of the cluster matches the observed state.

General pattern of a Kubernetes controller

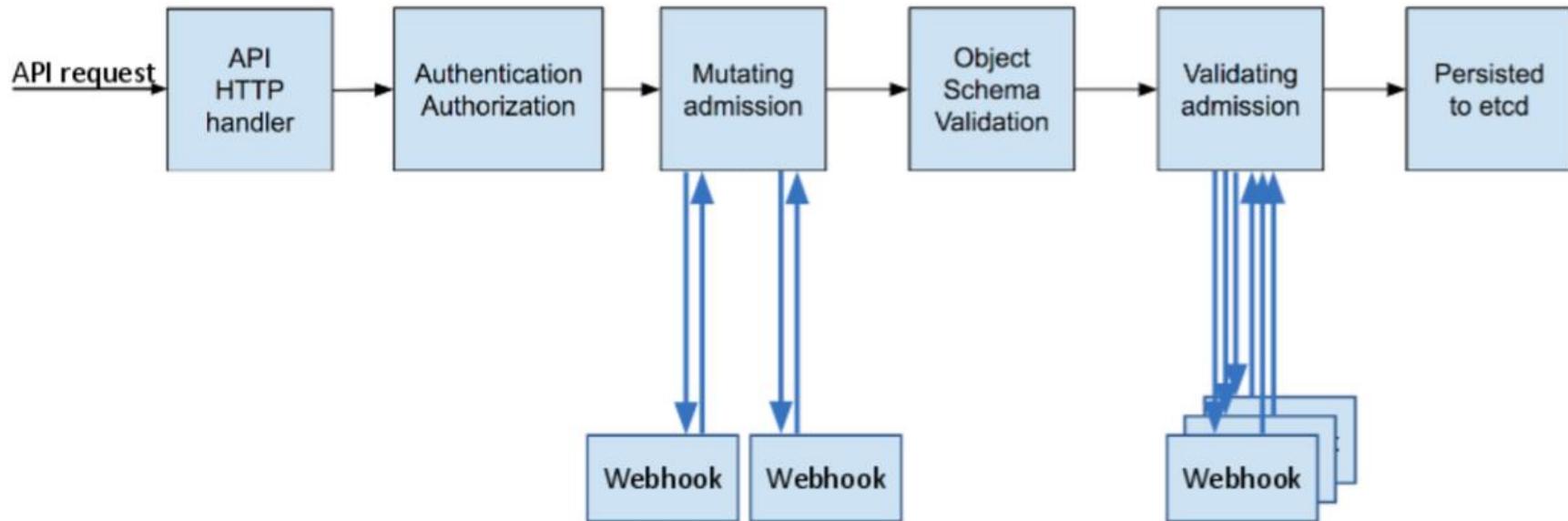


Controllers in Kubernetes

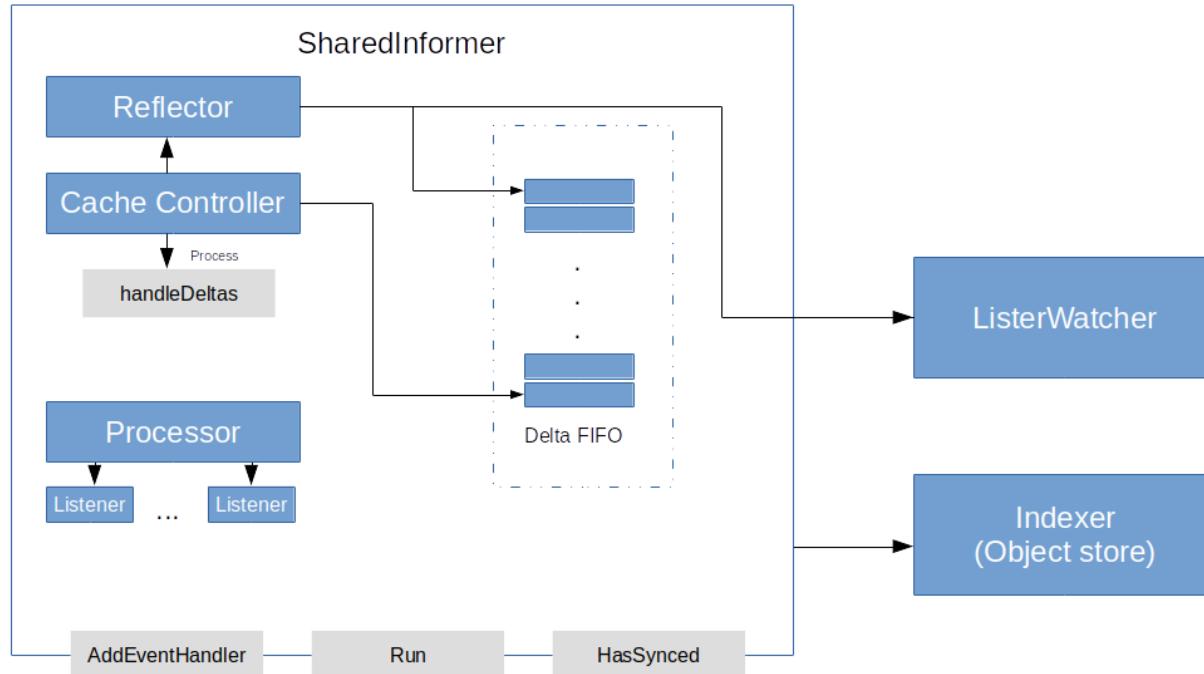
In Kubernetes, a controller is a control loop that watches the shared state of the cluster through the API server and makes changes attempting to move the current state towards the desired state. Examples of controllers that ship with Kubernetes today are the replication controller, endpoints controller, namespace controller, and service accounts controller.



Example: Admission Controller

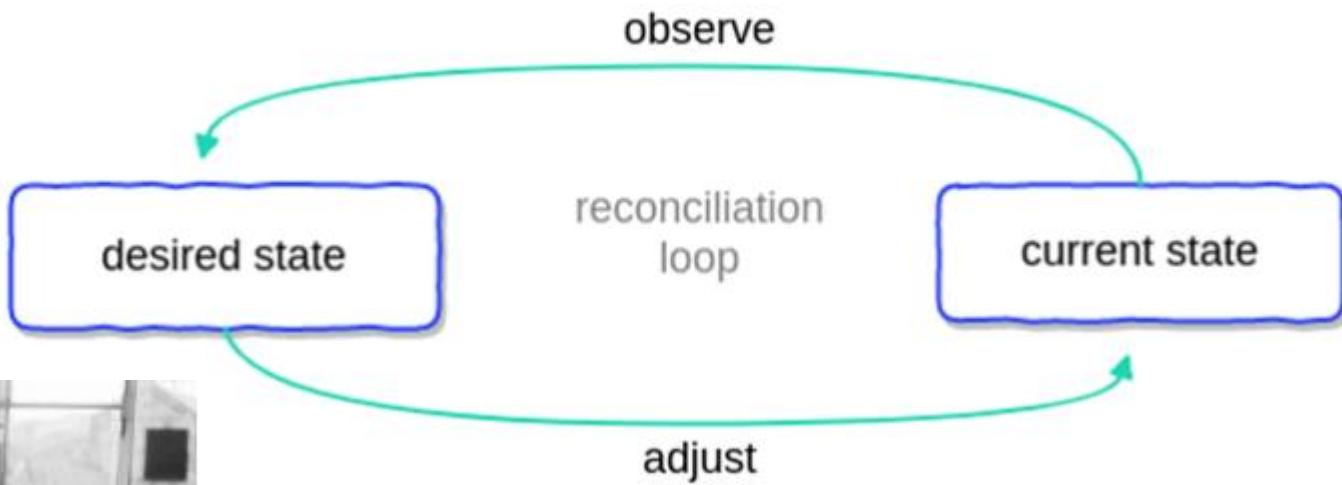


Controller Components

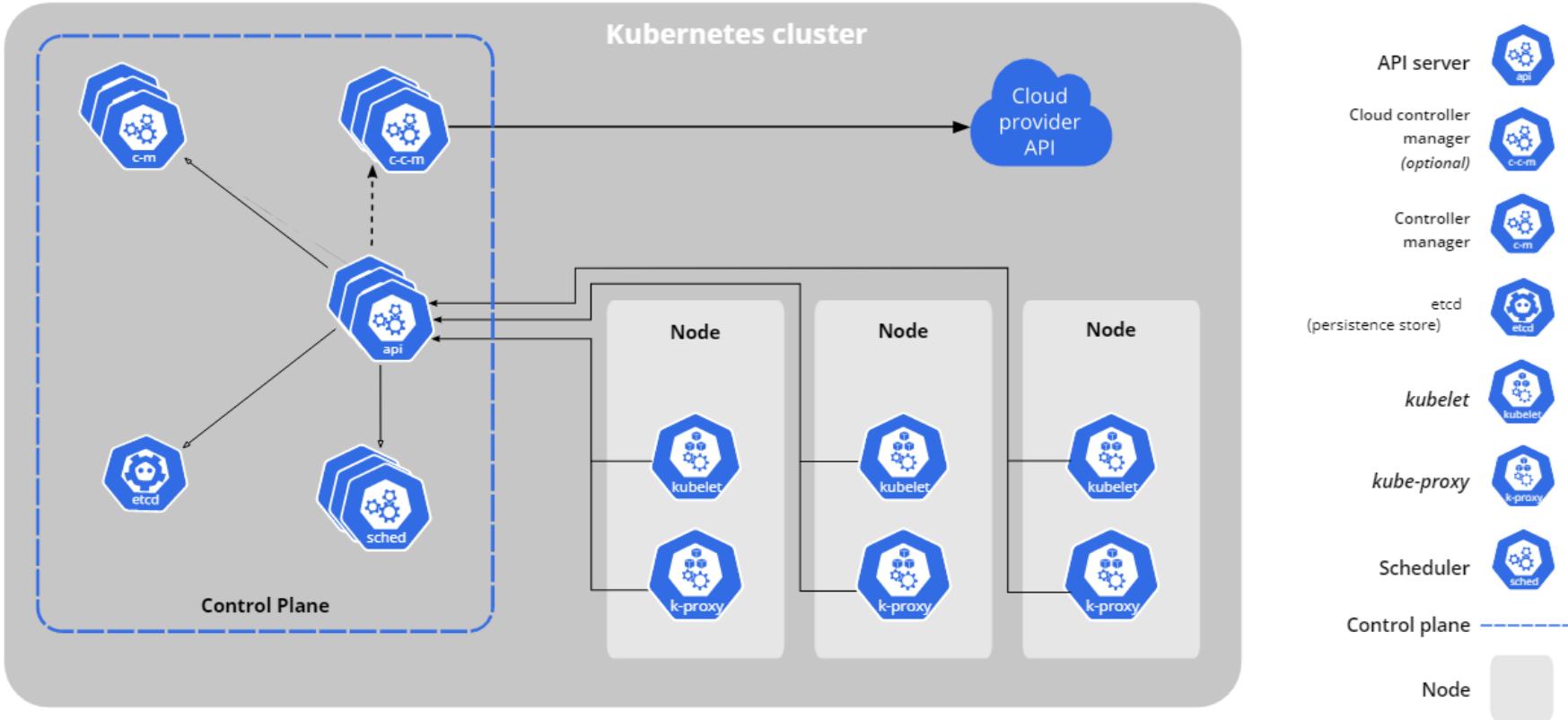


There are two main components of a controller: Informer/SharedInformer and Workqueue. Informer/Shared Informer watches for changes on the current state of Kubernetes objects and sends events to Workqueue where events are then popped up by worker(s) to process.

Controller loop



Controller Manager



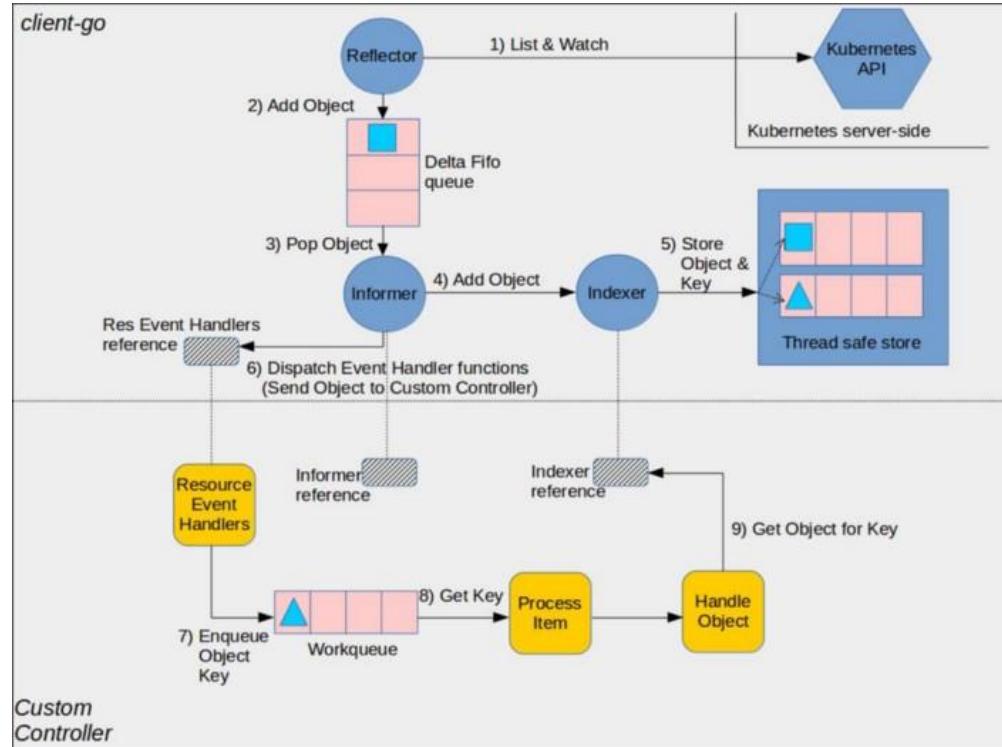
Node Controller



The controller manager is a daemon that is used for embedding core control loops, garbage collection, and Namespace creation. It enables the running of multiple processes on the master node even though they are compiled to run as a single process.

Creating Custom Controllers

The picture is divided into two parts— client-go and Custom Controller.



Building a Custom Controller to Manage Kubernetes Resources

Custom resources simply let you store and retrieve structured data. When you combine a custom resource with a *custom controller*, custom resources provide a true *declarative API*.

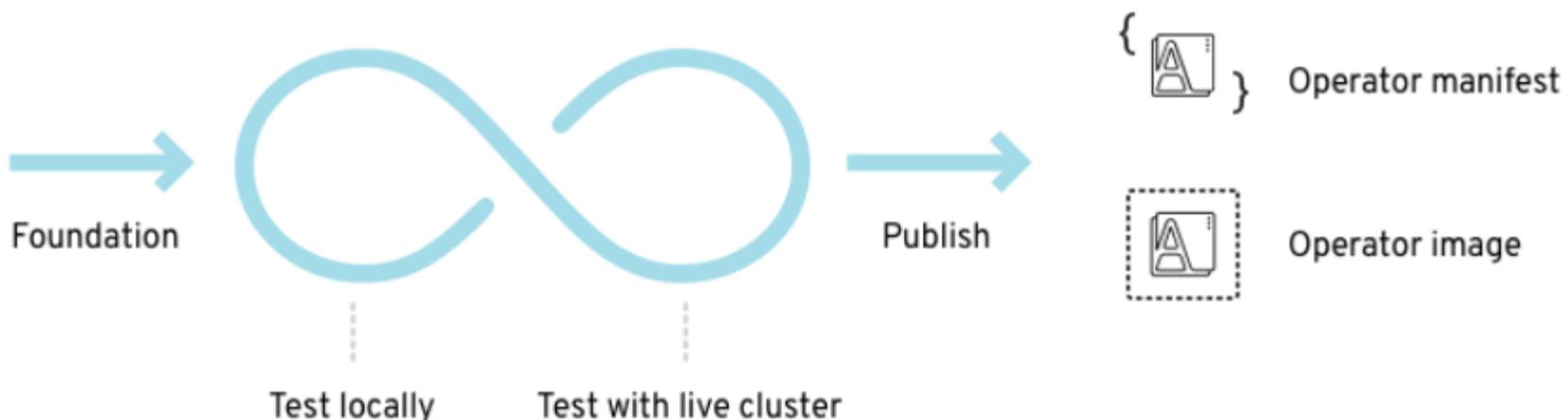
A [declarative API](#) allows you to *declare* or specify the desired state of your resource and tries to keep the current state of Kubernetes objects in sync with the desired state. The controller interprets the structured data as a record of the user's desired state, and continually maintains this state.



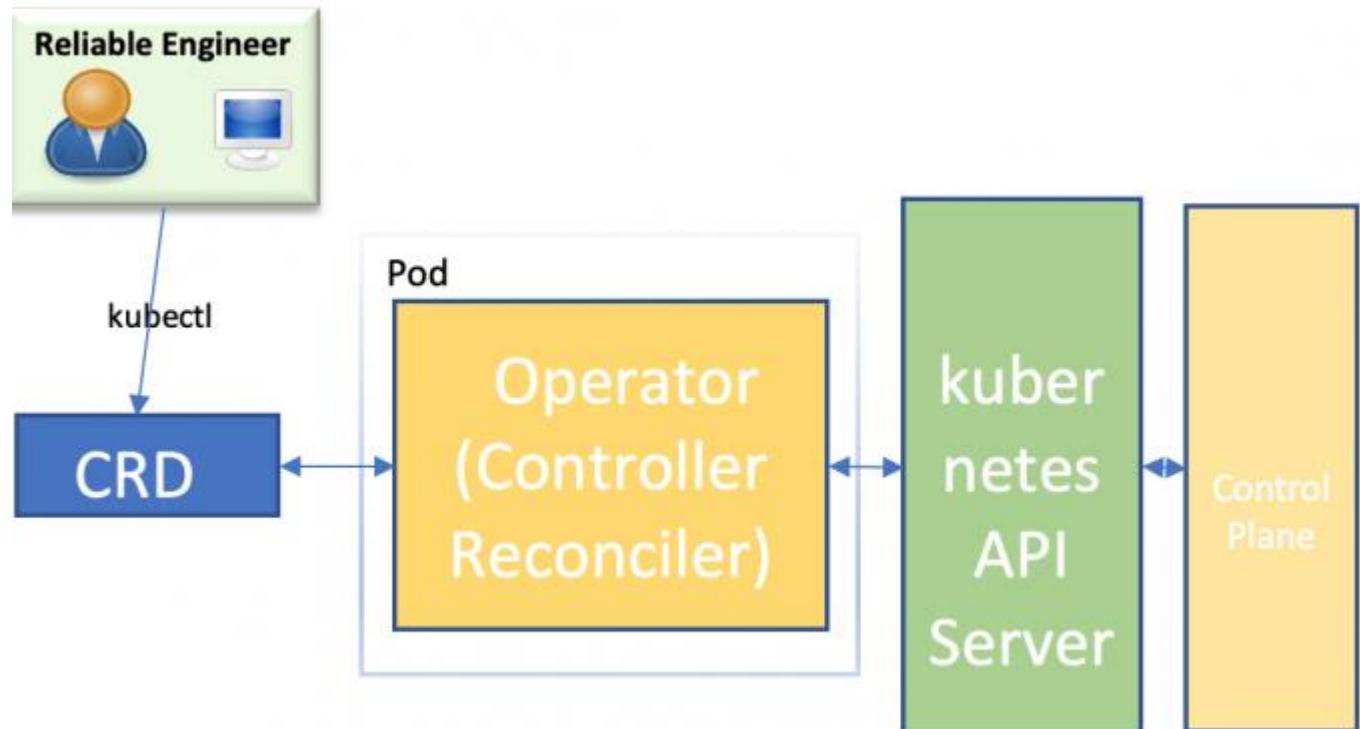
What is an Operator



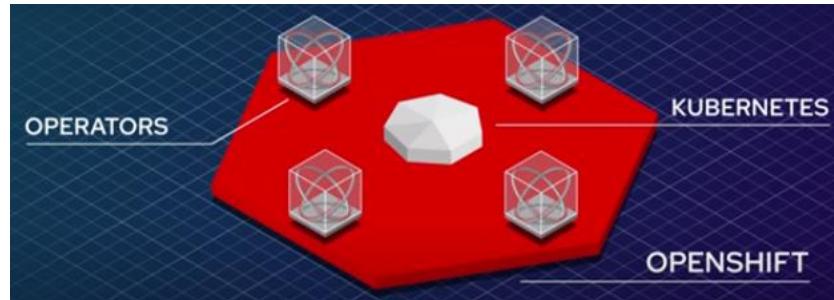
Operator SDK *Build, test, iterate*



Operator Pattern



Developing Operators



Phase I

Installation
Configuration and one-shot deployment

Phase II

Upgrades
Patch and minor version upgrades supported

Phase III

Lifecycle
App lifecycle, storage lifecycle (backup, failure recovery)

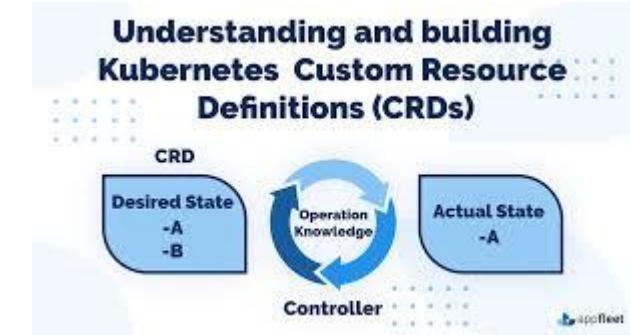
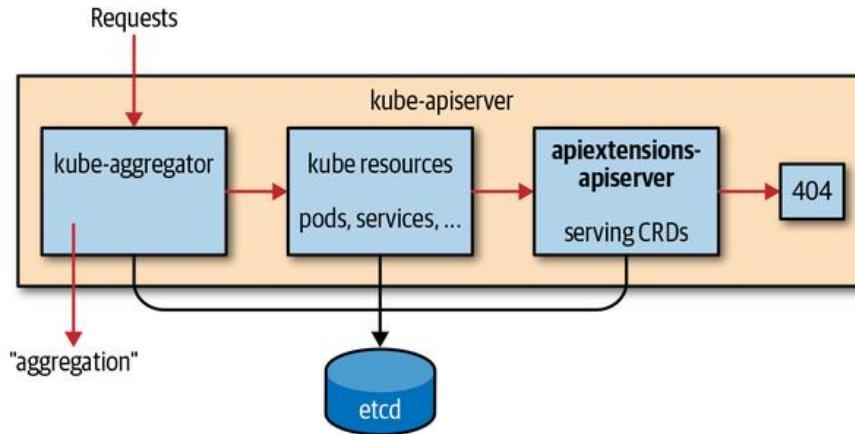
Phase IV

Insights
Metrics, alerts, log processing and workload analysis

Phase V

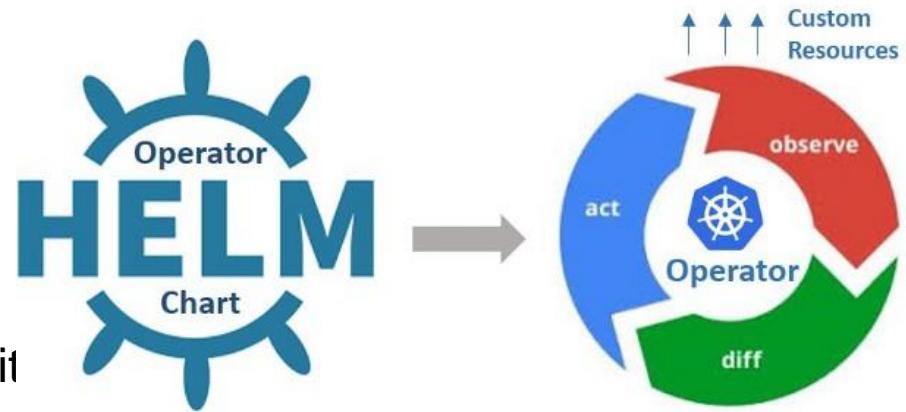
Auto-pilot
Horizontal/vertical scaling, auto config tuning, abnormal detection, scheduling tuning...

Building a Custom Resource Definition

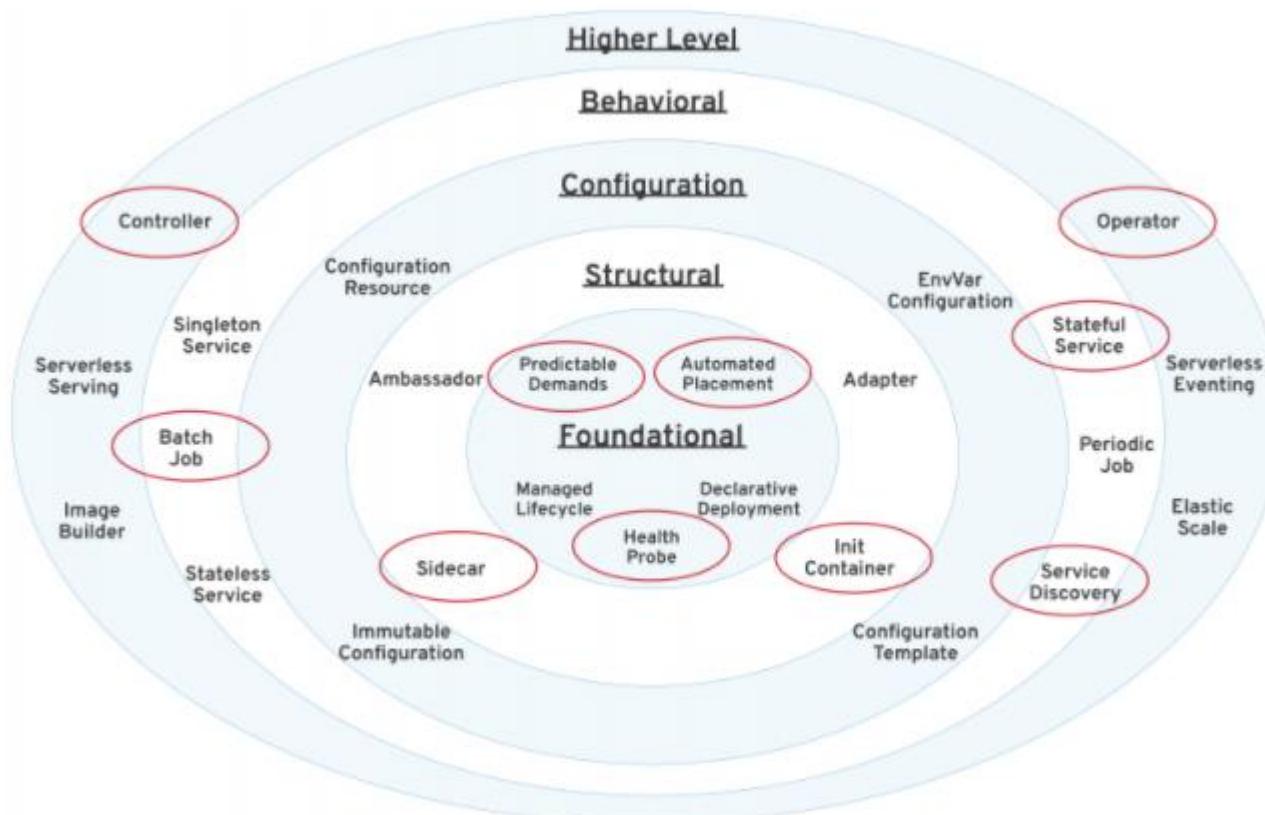


Operators vs Helm vs Controller

- Helm itself is an operator
- Controllers
 - Operator = controller + CRD
 - Operator = external software
 - Controller = internal
- Only do operators if you can't solve it with Helm



Application Patterns



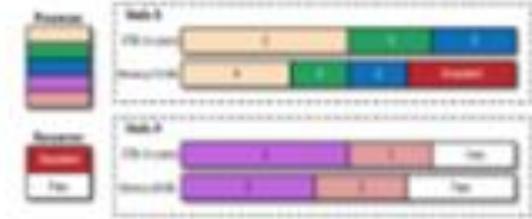
Foundational Application Patterns

Foundational Pattern

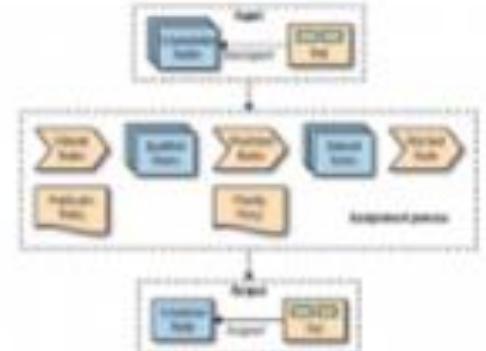
- Predictable demands
- Declarative deployment
- Health probe
- Managed lifecycle
- Automated placement



Health Probe



Predictable Demands

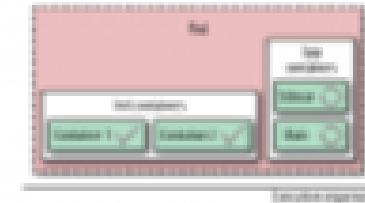


Automated Placement

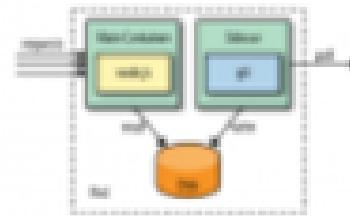
Behavioral Application Patterns

Behavioral
Pattern

- Batch job
- Periodic job
- Daemon service
- Stateful service
- Self-awareness



Init Container



Sidecar

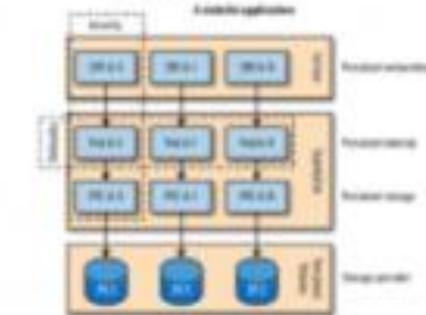
Structural Application Patterns

Structural Pattern

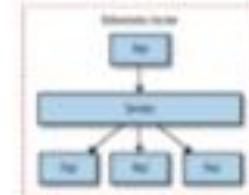
- Init containers
- Sidecar
- Adapter
- Ambassador



Batch Job



Stateful Service

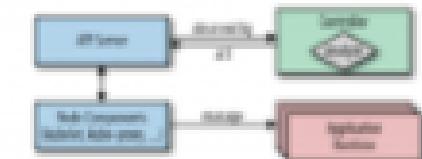


Service Discovery

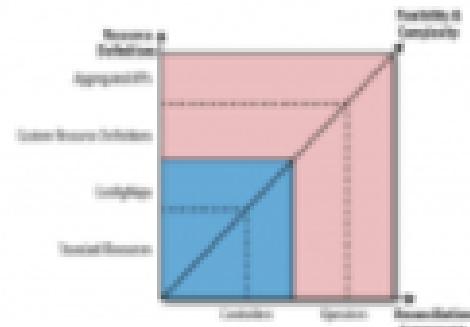
Higher-level Application Patterns

Higher-level Patterns

- Controller Pattern
- Operator Pattern



Controller



Operator

DevOps Application Patterns

- **Configuration patterns** are used for the various ways application configuration can be handled in Kubernetes.
- **Advanced patterns** include how the platform itself can be extended or how to build container images directly within the cluster.

POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



What are custom controllers?

A: patterns that sit on top of foundational patterns and add granularity to concepts for managing various types of container and platform interactions.

B: These patterns are the underlying principles and practices for building container-based cloud-native applications.

C: Patterns that are related to organizing containers within a Kubernetes pod.

POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



What is the Kubernetes Controller Manager?

- A: A set of machine elements that are nodes.
- B: A set of pods that run a set of machine elements that are nodes.ly once on a host.
- C: A daemon that is used for embedding core control loops, garbage collection, and Namespace creation

POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



What is the Kubernetes Controller Manager?

A: A set of machine elements that are nodes.

B: A set of pods that run a set of machine elements that are nodes.ly once on a host.

C: A daemon that is used for embedding core control loops, garbage collection, and Namespace creation

POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



Which of the following meets the definition of Foundational Patterns.

A: patterns that sit on top of foundational patterns and add granularity to concepts for managing various types of container and platform interactions.

B: These patterns are the underlying principles and practices for building container-based cloud-native applications.

C: Patterns that are related to organizing containers within a Kubernetes pod.

POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



Which of the following meets the definition of Foundational Patterns.

A: patterns that sit on top of foundational patterns and add granularity to concepts for managing various types of container and platform interactions.

B: These patterns are the underlying principles and practices for building container-based cloud-native applications.

C: Patterns that are related to organizing containers within a Kubernetes pod.

Operator SDK & Helm



Monitoring

reliable, resilient, observable



Observability & Monitoring

Observability is the property of the system that defines what we can tell about the state and behavior of the system, right now and historically.

- Gain understanding actively
- Ask questions based on hypotheses
- Build to tame dynamic environments with changing complexity
- Preferred by developers of systems with variability and unknown permutations

Monitoring is the collection of tools, processes, and techniques we use to increase the observability of the system.

- Consume information passively
- Ask questions based on dashboards
- Built to maintain based on dashboards

Built to maintain static environments with little variation

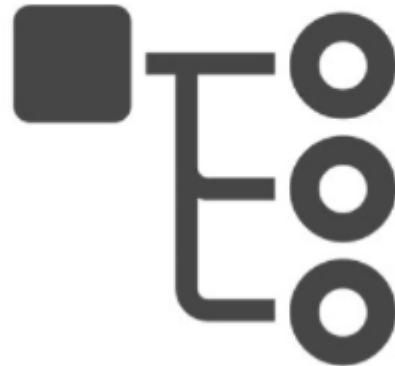
Used by developers of systems with little change and known permutations

Observability

Three pillars of observability



Metrics



Traces



Logs



Why?



Rise of Prometheus

The following diagram illustrates the entire system:

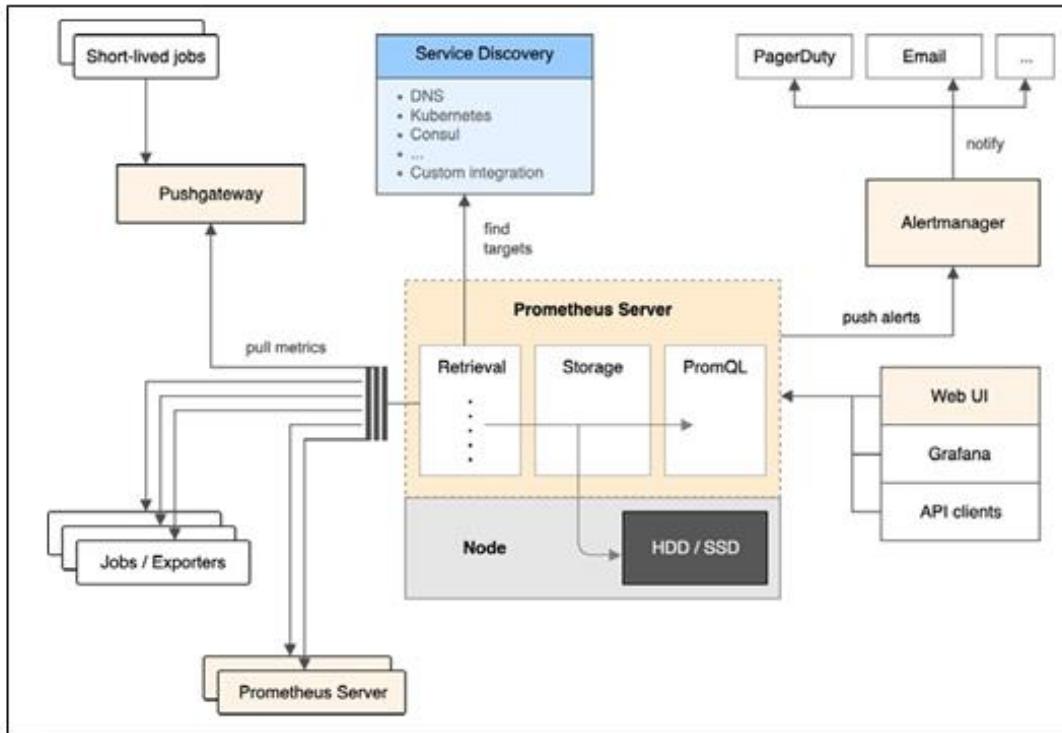
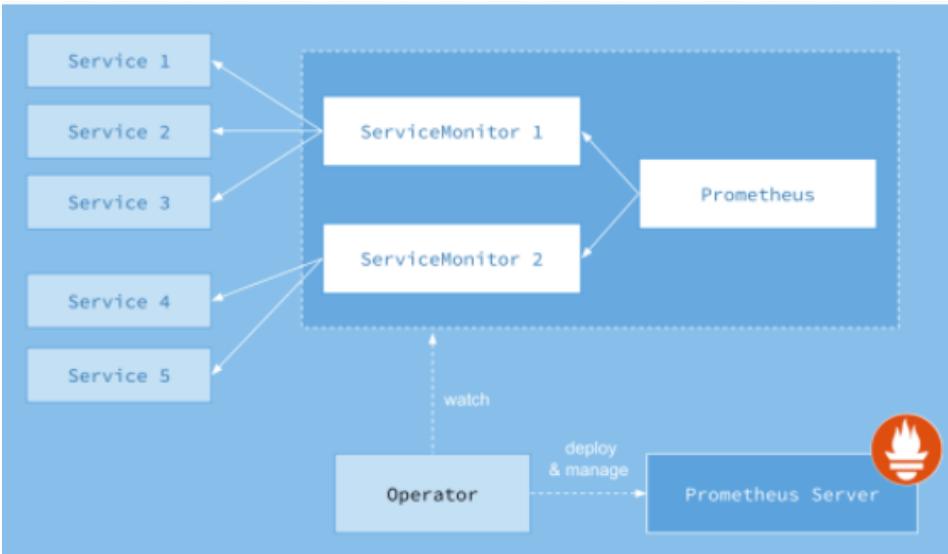


Figure 13.7: the Prometheus system

Installing Prometheus

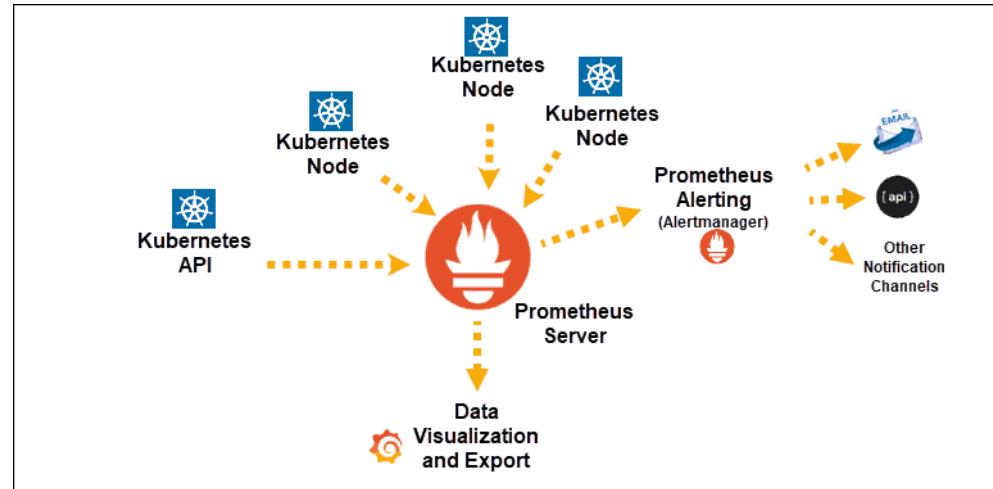


Monitoring with Prometheus

Why Use Prometheus for Kubernetes Monitoring

Two technology shifts took place that created a need for a new monitoring framework:

1. DevOps culture
2. Containers and Kubernetes



Interacting With Prometheus

Prometheus has a basic web UI that you can use to explore its metrics. Let's do port forwarding to localhost:

```
$ POD_NAME=$(kubectl get pods -n monitoring -l "app=prometheus" \
    -o jsonpath=".items[0].metadata.name")  
  
$ kubectl port-forward -n monitoring $POD_NAME 9090
```

Then, you can browse to

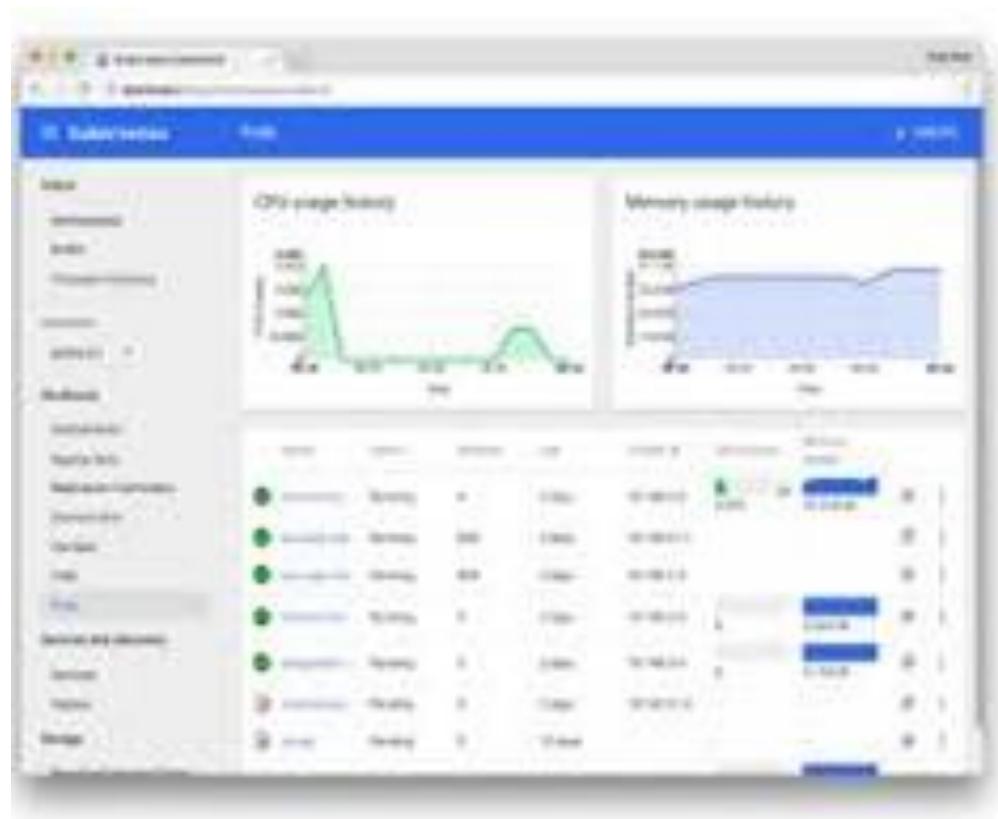
<http://localhost:9090>, where you can select different metrics and view raw data or graphs:

Prometheus records an outstanding number of metrics (990, in my current setup). The most relevant metrics on Kubernetes are the metrics exposed by kube-state-metrics and node exporters.

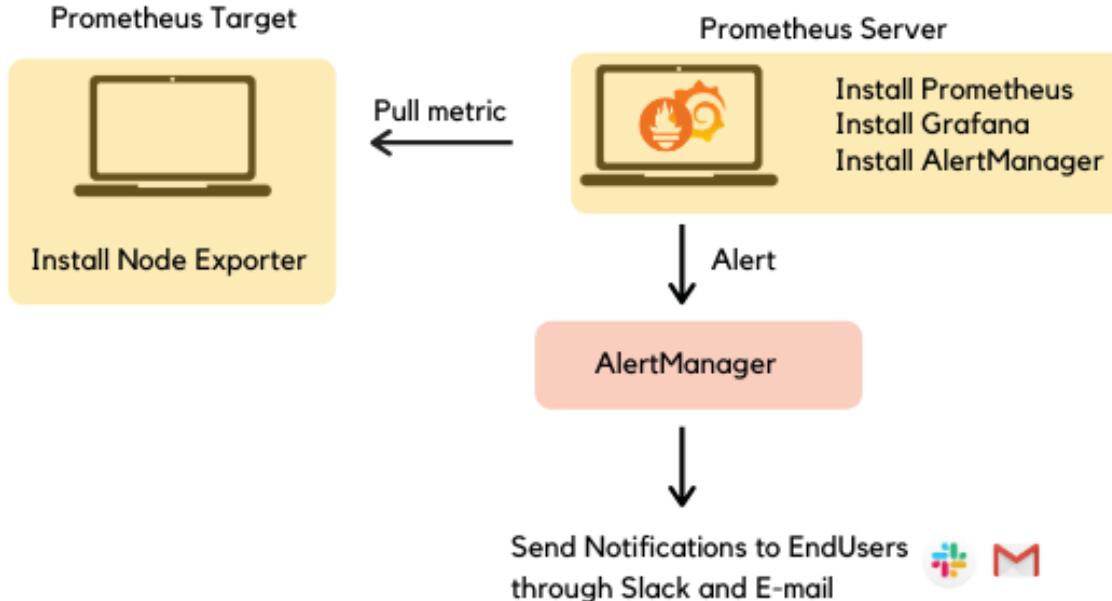


Incorporating Kube-state-metrics

The Prometheus operator already installs kube-state-metrics. It is a service that listens to Kubernetes events and exposes them through a /metrics HTTP endpoint in the format that Prometheus expects.



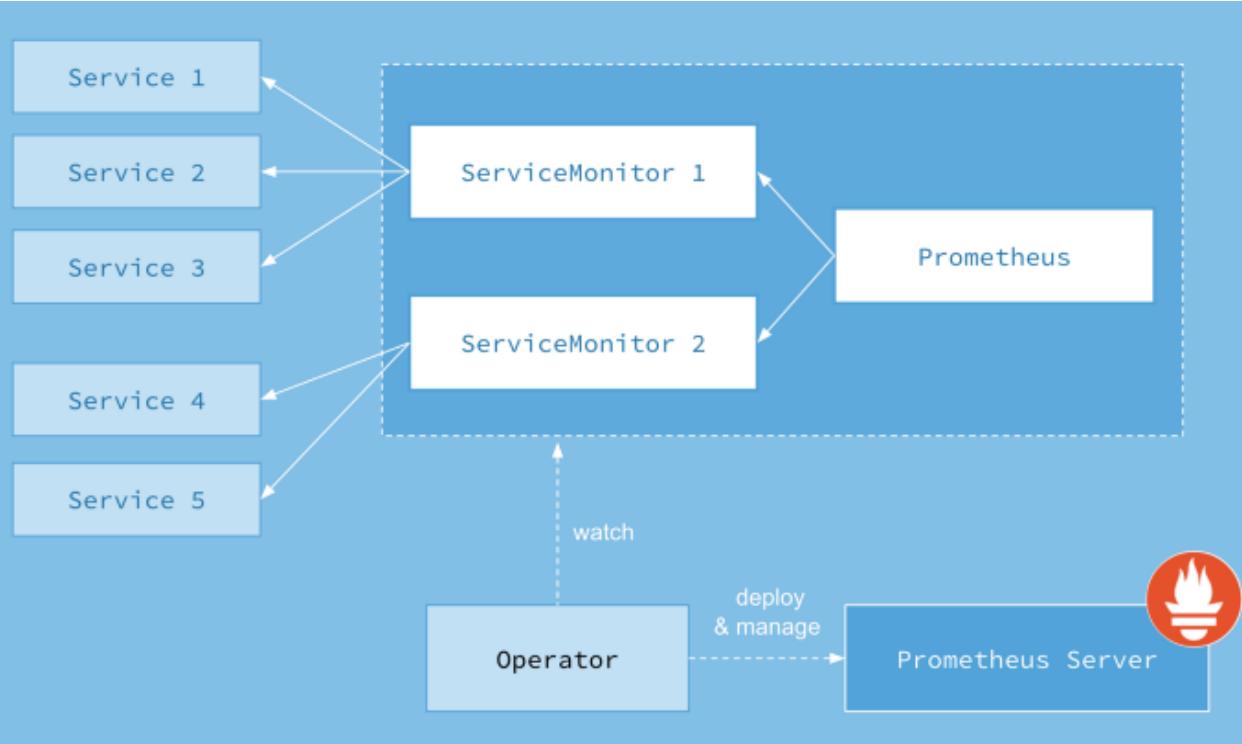
Triggering Alerts with AlertManager





Prometheus

Prometheus Operator



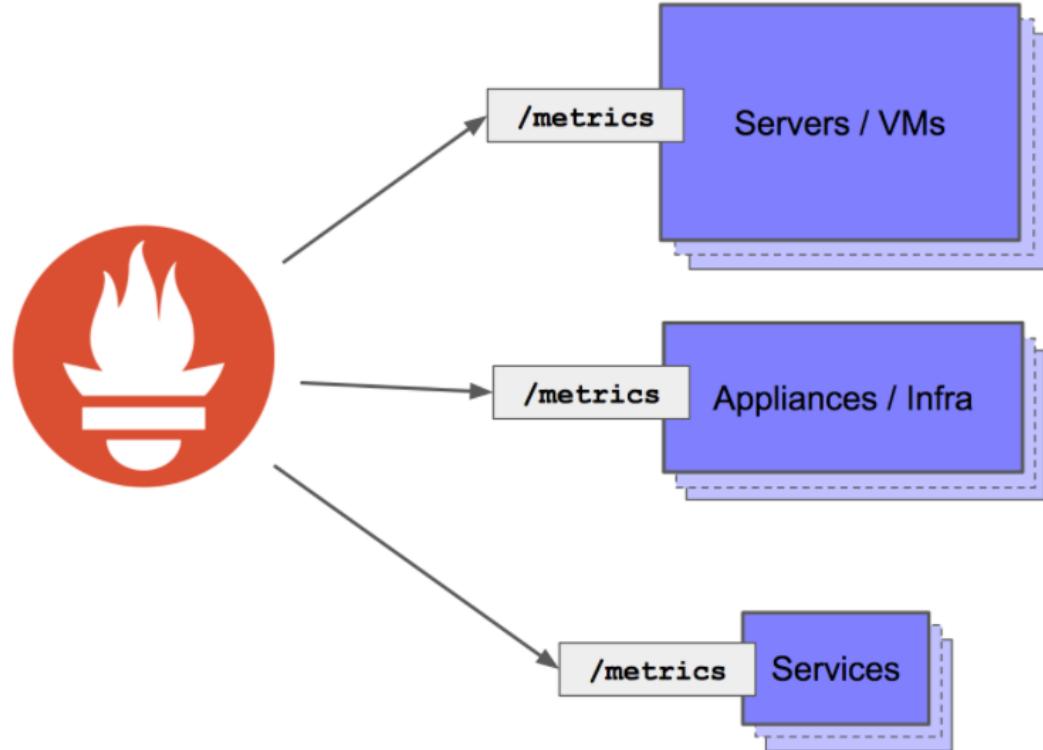
The Prometheus Operator provides easy monitoring for k8s services and deployments besides managing Prometheus, Alertmanager and Grafana configuration.



Prometheus Operator

- **Kubernetes Custom Resources:** Use Kubernetes custom resources to deploy and manage Prometheus
- **Simplified Deployment Configuration:** Configure the fundamentals of Prometheus like versions, persistence, retention policies, and replicas
- **Prometheus Target Configuration:** Automatically generate monitoring target configurations based on Kubernetes label queries

Exposing Metrics with Prometheus



Deploying Prometheus Using Helm

1. Update the Helm repository. This command will fetch up-to-date charts locally from public chart repositories:

```
$ helm repo update
```

2. Deploy Prometheus Operator in the monitoring namespace using the `helm install` command. This command will deploy Prometheus along with the Alertmanager, Grafana, the node-exporter and kube-state-metrics addon; basically, a bundle of the components needed to use Prometheus on a Kubernetes cluster:

```
$ helm install stable/prometheus-operator --name prometheus \
--namespace monitoring
```

3. Verify the status of the pods deployed in the monitoring namespace:

```
$ kubectl get pods -n monitoring
NAME READY STATUS RESTARTS AGE
alertmanager-prometheus-prometheus-oper-alertmanager-0 2/2 Running
0 88s
prometheus-grafana-6c6f7586b6-f9jbr 2/2 Running 0 98s
prometheus-kube-state-metrics-57d6c55b56-wf4mc 1/1 Running 0 98s
prometheus-prometheus-node-exporter-8drg7 1/1 Running 0 98s
prometheus-prometheus-node-exporter-lb715 1/1 Running 0 98s
prometheus-prometheus-node-exporter-vx7w2 1/1 Running 0 98s
prometheus-prometheus-oper-operator-86c9c956dd-88p82 2/2 Running 0
98s
prometheus-prometheus-prometheus-oper-prometheus-0 3/3 Running 1
78s
```



Prometheus

Node Exporter

`kube-state-metrics` collects node information from the Kubernetes API server, but this information is pretty limited.

Prometheus comes with its own node exporter, which collects tons of low-level information about the nodes.

Remember that Prometheus may be the de facto standard metrics platform on Kubernetes, but it is not Kubernetes-specific.

Here is a small subset of the metrics exposed by the node exporter:

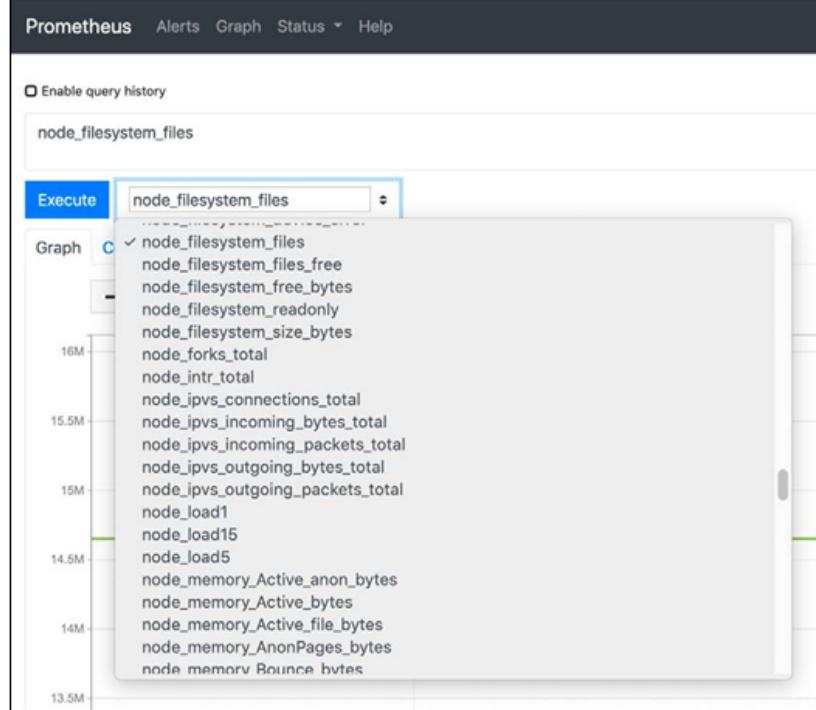


Figure 13.9: Metrics exposed by the node exporter

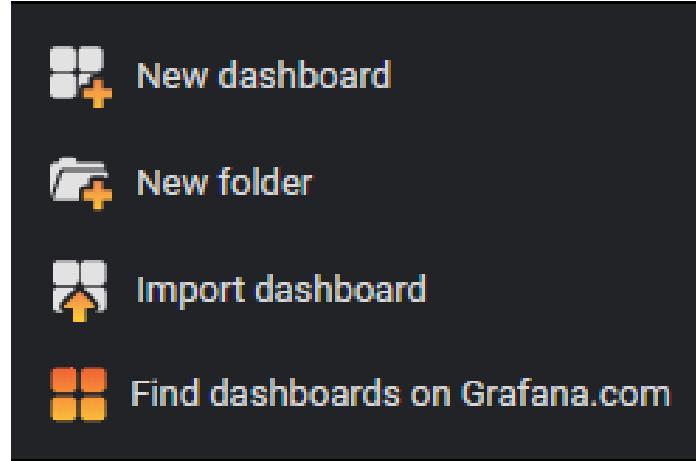
Grafana

Grafana is an open source analytics and monitoring solution. By default, Grafana is used for querying Prometheus. Follow these instructions to expose the included Grafana service instance and access it through your web browser:

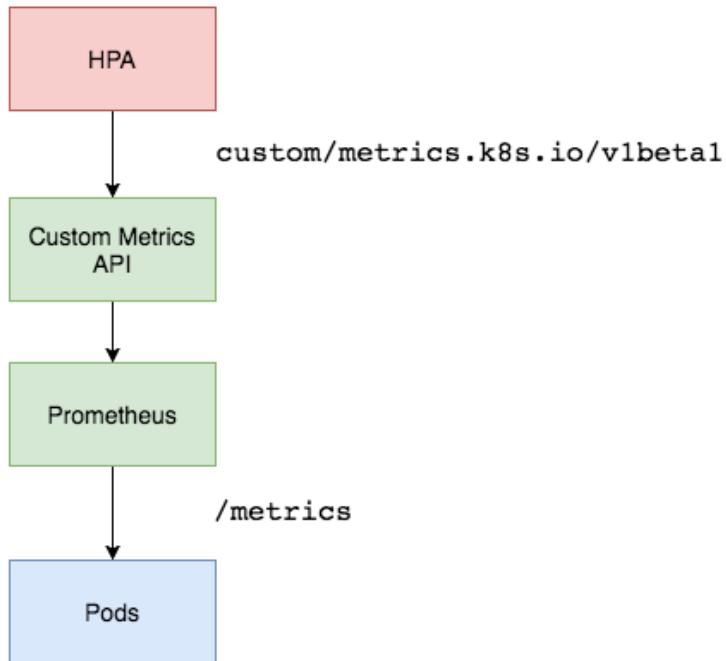


Grafana Dashboard

Grafana is an open sourced monitor used to visualize the metrics stored on Prometheus. It offers dynamic and reusable dashboards with template variables. In this recipe, we will learn how to add a new dashboard from the library of pre-built dashboards to monitor an application deployed on Kubernetes.



Creating Custom Metrics to Trigger Cluster Autoscaling



Logging

Logging is a key monitoring tool. Every self-respecting long-running software must have logs. Logs capture timestamped events. They are critical for many

applications, like business intelligence, secu



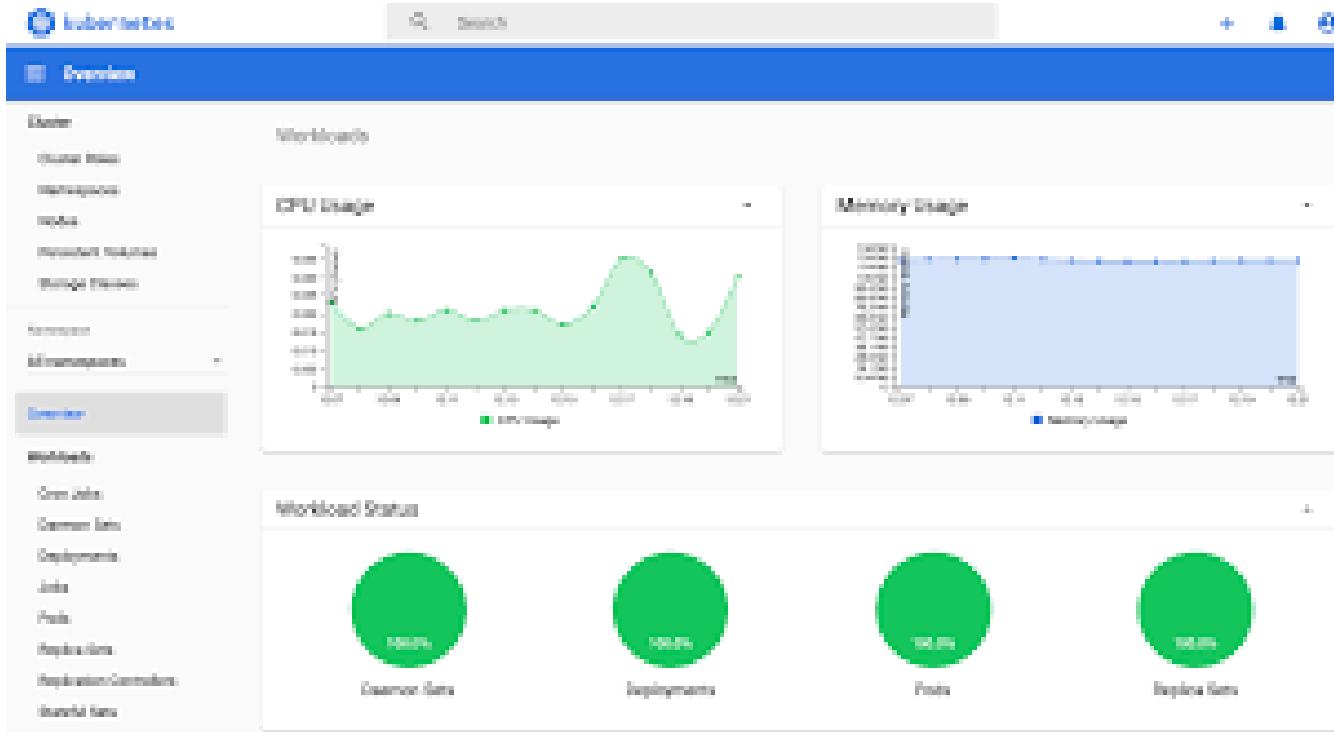
Metrics

Metrics measure the same aspect of the system over time. Metrics are time series of numerical values (typically, floating-point numbers). Each metric has a name and often a set of labels that help later in slicing and dicing. For example, the CPU utilization of a node or the error rate of a service are metrics.



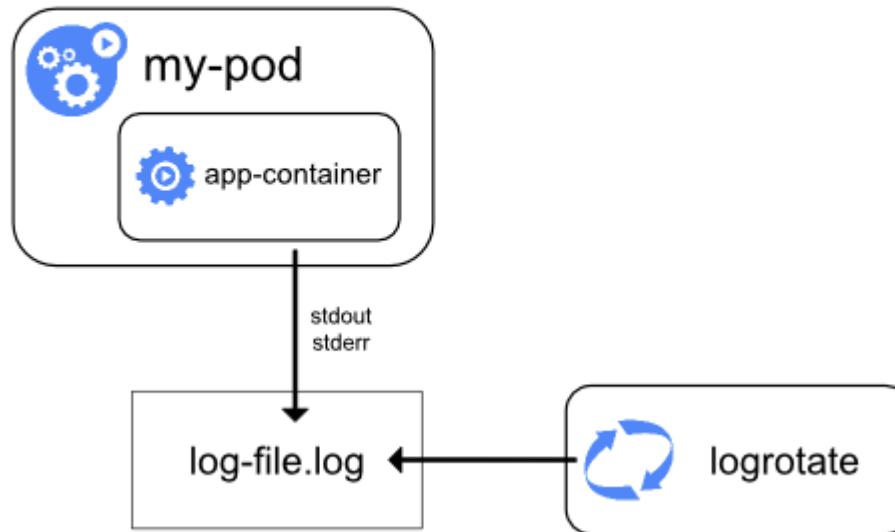
Monitoring Metrics using Kubernetes Dashboard

Kubernetes Dashboard doesn't display detailed metrics unless Kubernetes Metrics Server is installed and the kubernetes-metrics-scraper sidecar container is running



Logging Solution

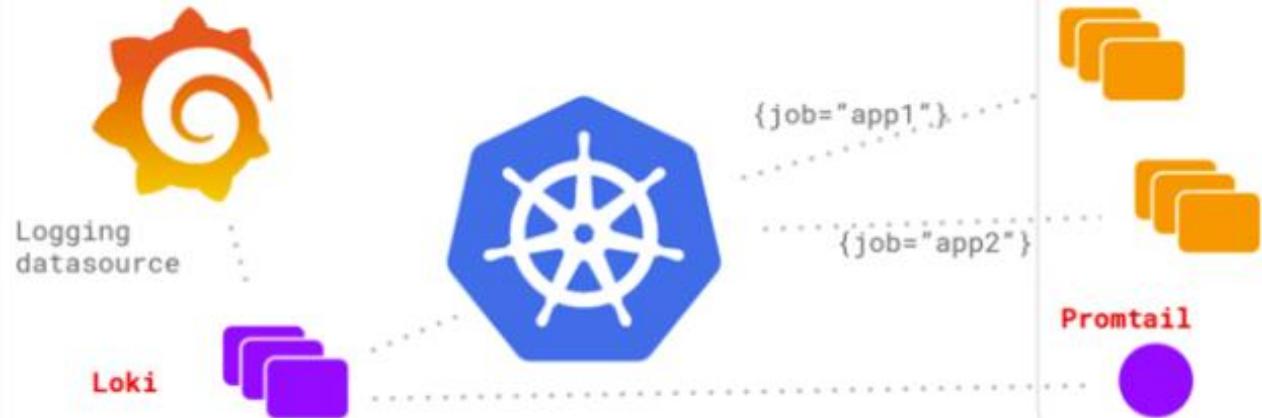
Logging at the node level



Loki

Loki is a logging backend optimized for users running Prometheus and Kubernetes.

Logging architecture



Loki was built for efficiency alongside the following goals:

- Logs should be cheap. Nobody should be asked to log less.
- Easy to operate and scale.
- Metrics, logs (and traces later) need to work together.

Logging with Loki

2019-12-11T10:01:02.123456789Z {app="nginx", instance="1.1.1.1"} GET /about

Timestamp

with nanosecond precision

Prometheus-style **Labels**

key-value pairs

Content

log line

indexed

unindexed

Selecting Log Streams

Selecting logstreams, is done by using **label matchers** and **filter expressions**

Log stream

{app="nginx", instance="1.1.1.1"}

Query: {app="nginx"} start=T5 end=T7

{app="nginx", instance="2.2.2.2"}

Chunks

T1-T5

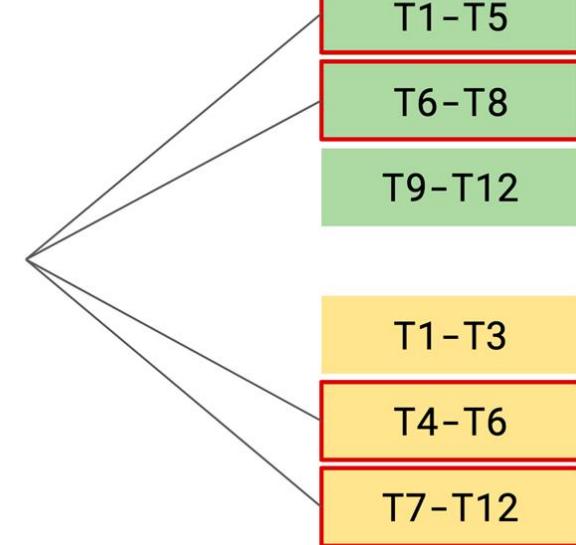
T6-T8

T9-T12

T1-T3

T4-T6

T7-T12



POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



What are the three pillars of observability?

- A: Metrics, Traces, Logs
- B: Technique, Process, Metrics
- C: Efficiency, Metrics, Monitoring



POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



What are the three pillars of observability?

- A: Metrics, Traces, Logs
- B: Technique, Process, Metrics
- C: Efficiency, Metrics, Monitoring

POP QUIZ:

Kubernetes: Controller, Operator Patterns, Creation



What is the difference between Grafana and Prometheus?

A: Grafana is an open-source visualization software, which helps the users to understand the complex data with the help of data metrics. Prometheus is an open-source event monitoring and alerting tool. It stores the majority of the data locally after scrapping metrics

B: Prometheus Is an open-source visualization software, which helps the users to understand the complex data with the help of data metrics. Grafana is an open-source event monitoring and alerting tool. It stores the majority of the data locally after scrapping metrics

POP QUIZ:

KUBERNETES DEVOPS



What is the difference between Grafana and Prometheus?

A: Grafana is an open-source visualization software, which helps the users to understand the complex data with the help of data metrics. Prometheus is an open-source event monitoring and alerting tool. It stores the majority of the data locally after scrapping metrics

B: Prometheus is an open-source visualization software, which helps the users to understand the complex data with the help of data metrics. Grafana is an open-source event monitoring and alerting tool. It stores the majority of the data locally after scrapping metrics

POP QUIZ:

KUBERNETES DEVOPS



What is the mission of Prometheus Operator?

- A: To make running prometheus as efficient as possible
- B: To make running prometheus as fast as possible
- C: To make running Prometheus on top of Kubernetes as easy as possible



POP QUIZ:

KUBERNETES DEVOPS



What is the mission of Prometheus Operator?

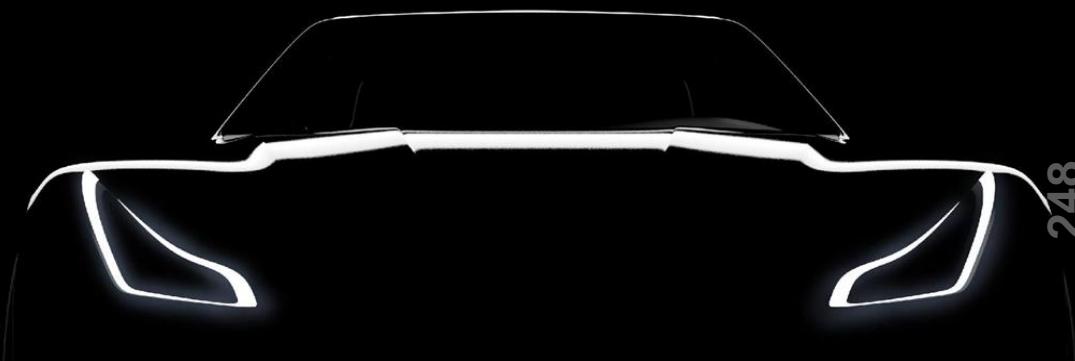
- A: To make running prometheus as efficient as possible
- B: To make running prometheus as fast as possible

C: To make running Prometheus on top of Kubernetes as easy as possible

Experiment – Jaeger Operator



Service Mesh



What is a Service Mesh

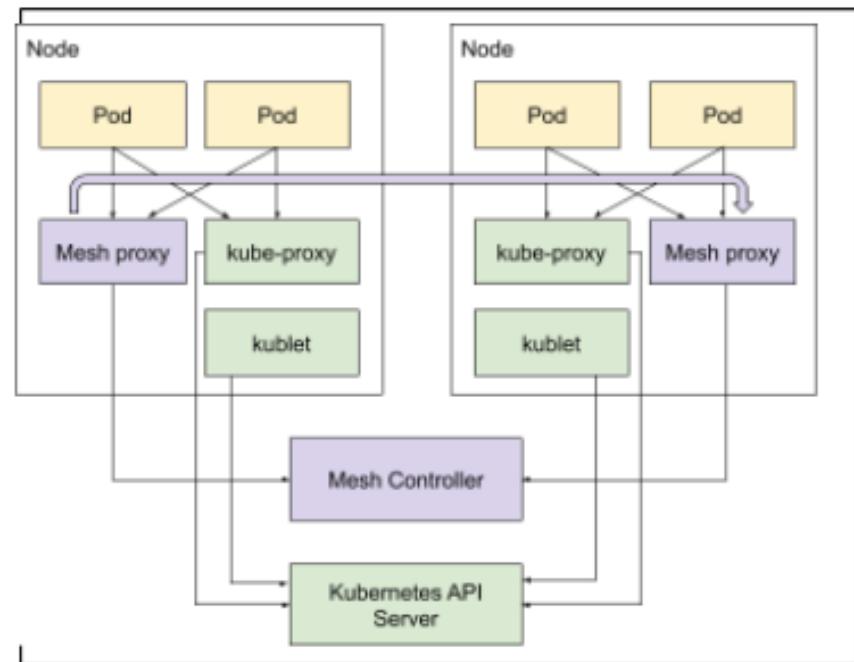
A service mesh is a dedicated infrastructure layer for handling service-to-service communication.

The components:

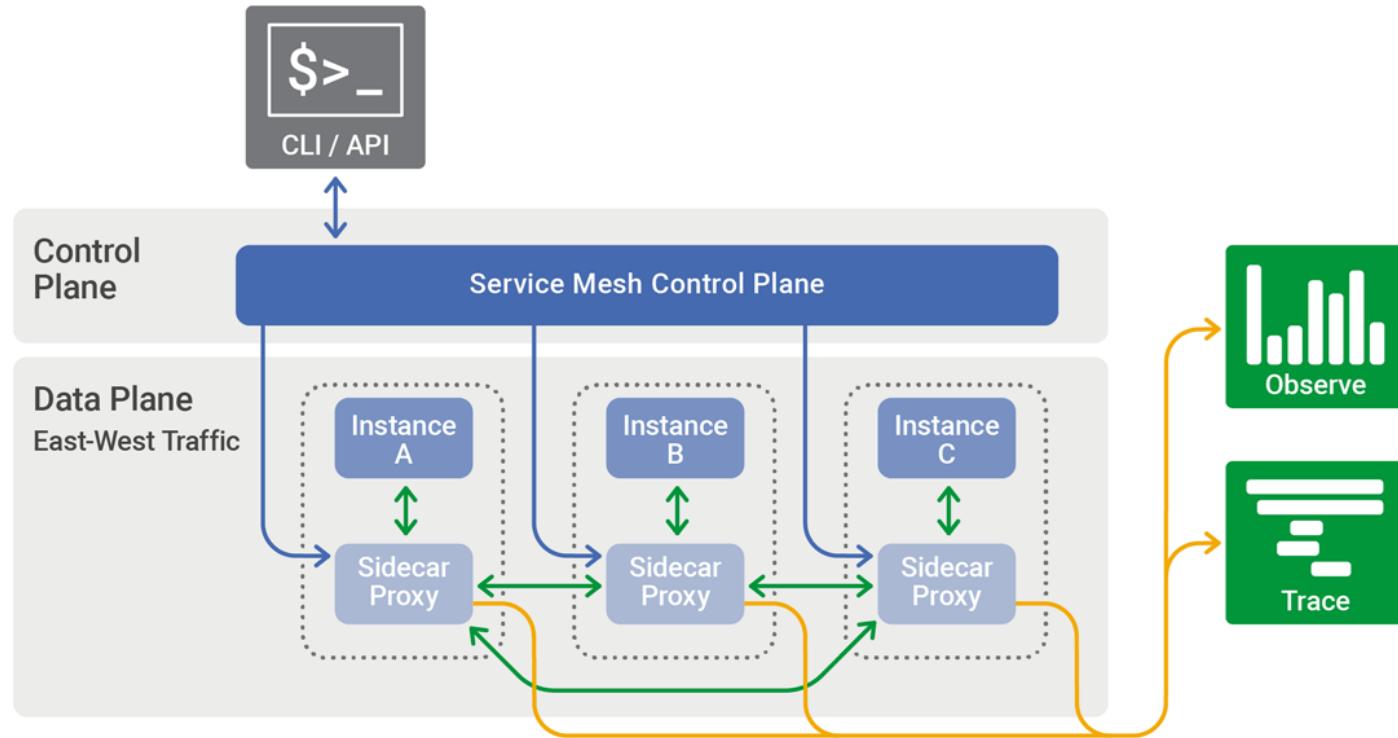
Data plane - lightweight proxies that are distributed as sidecars. Proxies include NGINX, or envoy

Control plane - provides the configuration for the proxies, issues the TLS certificates authority, and contain the policy managers.

The following diagram illustrates how a service mesh is embedded into a Kubernetes cluster:



What is a Service Mesh



Why Service Mesh?

- Advanced load balancing
- Service discovery
- Support canary deployments
- Caching
- Tracing a request across multiple microservices
- Authentication between services
- Throttling the number requests a service handles at a given time
- Automatically retrying failed requests
- Failing over to an alternative component when a component fails consistently
- Collecting metrics

Optimizing Communication

a service mesh also captures every aspect of service-to-service communication as performance metrics. Over time, data made visible by the service mesh can be applied to the rules for interservice communication, resulting in more efficient and reliable service requests.

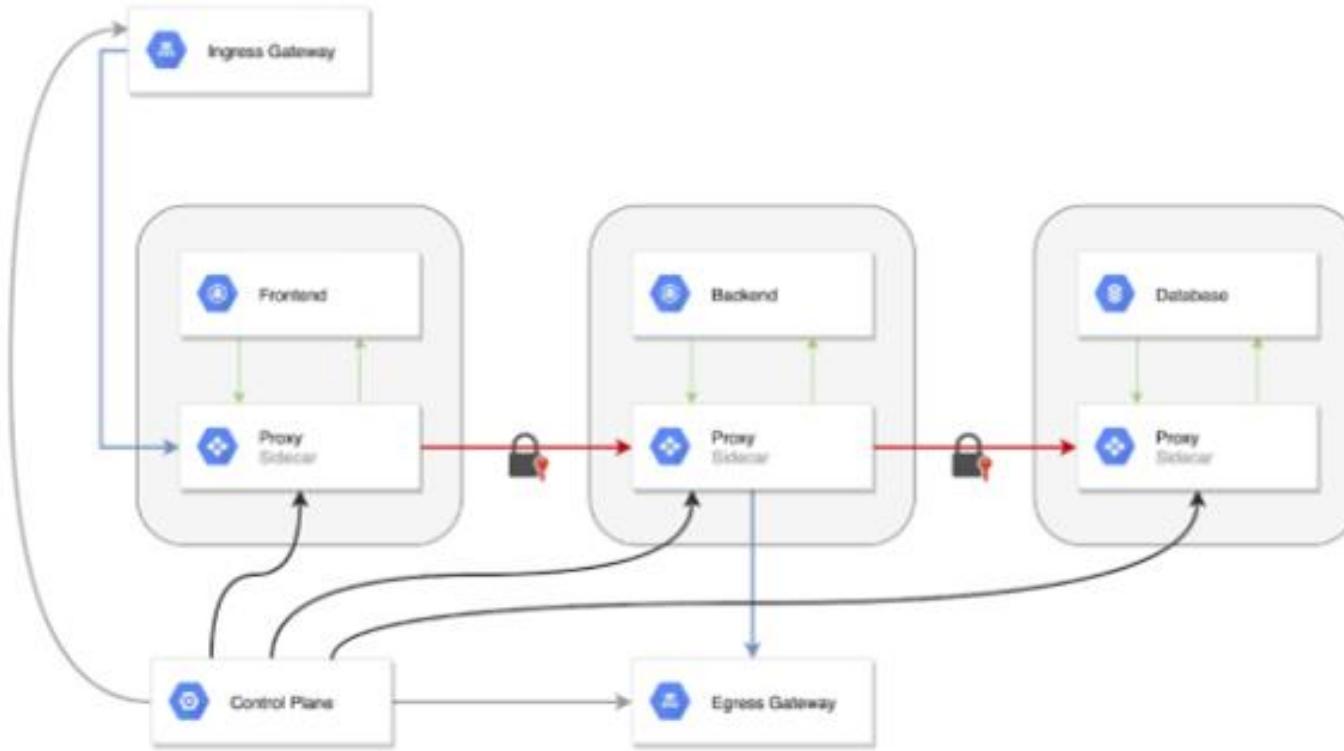


Observability

Many organizations are initially attracted to the uniform observability that service meshes provide. No complex system is ever fully healthy. Service-level telemetry illuminates where your system is behaving sickly, illuminating difficult-to-answer questions like why your requests are slow to respond



Kubernetes and Service Mesh



Implementations

Istio

AWS App Mesh

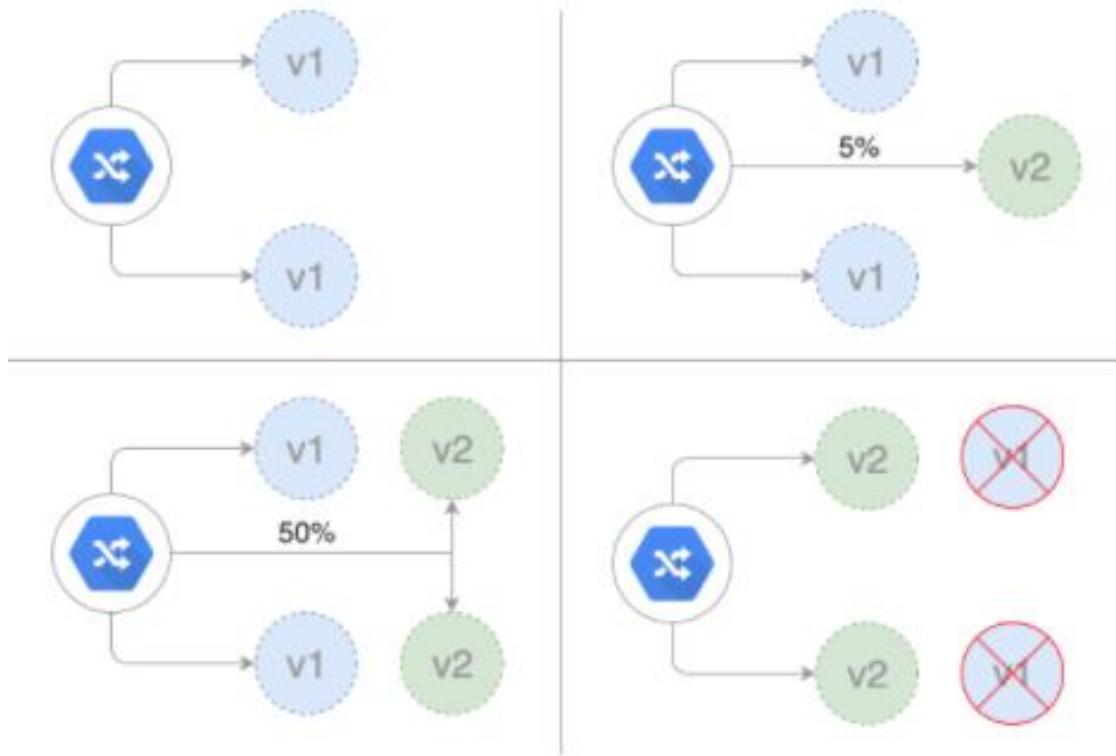
Linkerd v2

Consul Connect

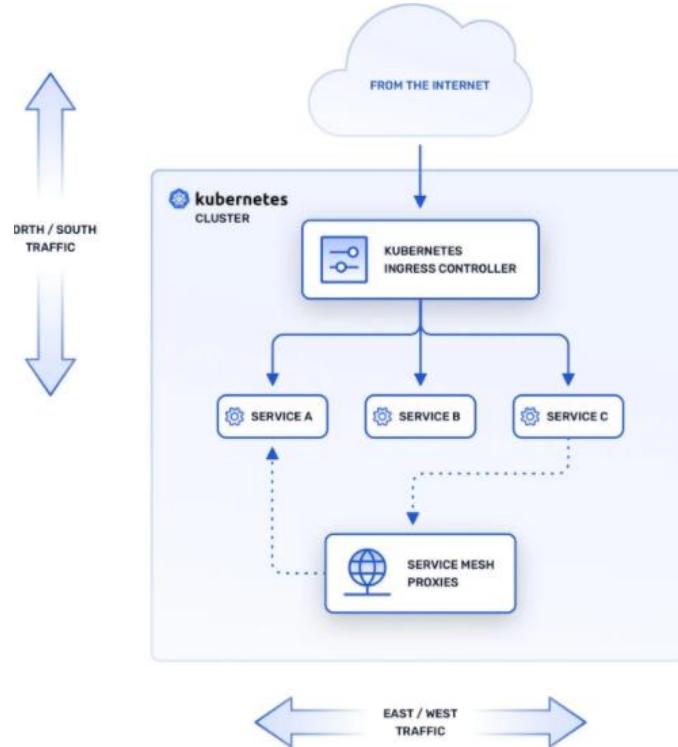


Canary

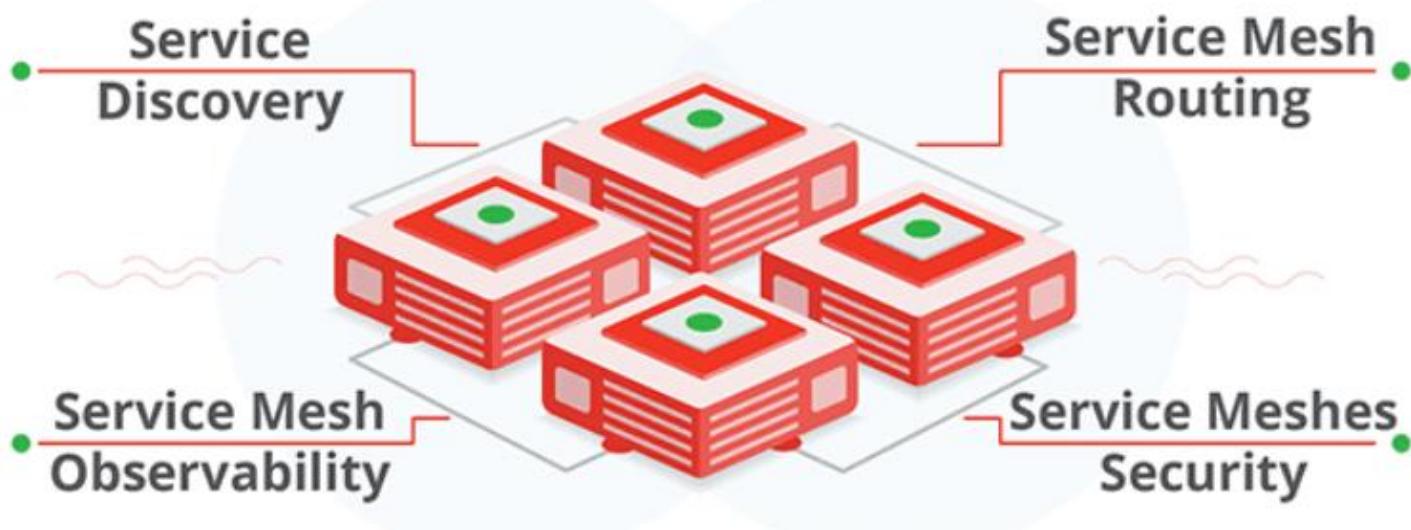
A **canary** is used for when you want to test some new functionality typically on the backend of your application.



Problem Statement



Architecture



Best Practices

- Auto-inject proxies
- Focus on automation
- Monitor and Trace everything



Implementing Retry Logic

Implement retry logic only where the full context of a failing operation is understood. For example, if a task that contains a **retry** policy invokes another task that also contains a **retry** policy, this extra layer of **retries** can add long delays to the processing

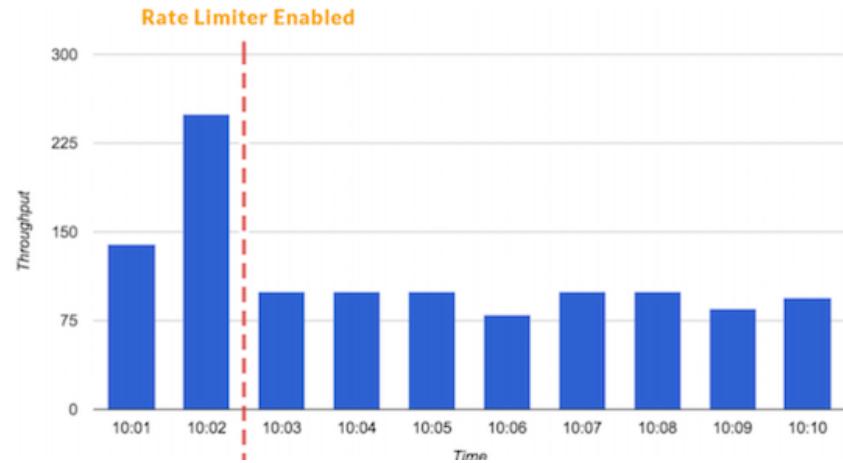
Object	API	Version
VirtualService	networking.istio.io	v1alpha3



Rate limiters & Load Shredders

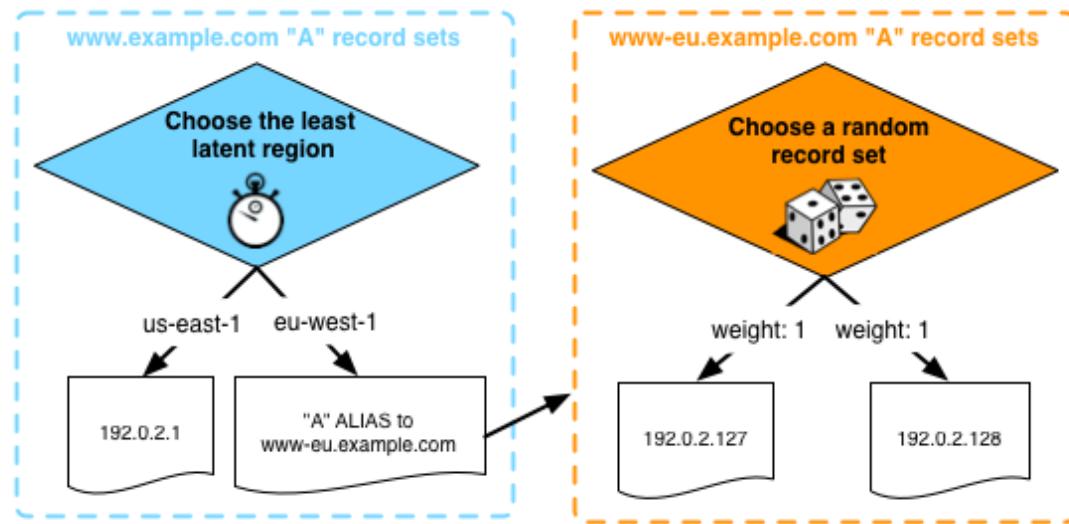
Rate limiting is the technique of defining how many requests can be received or processed by a particular customer or application during a timeframe

A *fleet usage load shredder* can ensure that there are always enough resources available to serve critical transactions. It keeps some resources for high priority requests and doesn't allow for low priority transactions to use all of them



Implementing Weighted Routing

Weighted routing lets you associate multiple resources with a single domain name (example.com) or subdomain name (acme.example.com) and choose how much traffic is routed to each resource.



Implementing Access Controls

tips for implementing access control systems:

1. Implement a central repository with well-defined whitelisting policies. ...
2. Solve self-generated scripts. ...
3. Withdraw your departing employees' digital rights. ...
4. Adapt your **access control**. ...
5. Create consistent processes to whitelist new cloud applications



POP QUIZ:

KUBERNETES DEVOPS



What is service mesh?

- A: a dedicated infrastructure layer for handling service-to-service communication.
- B: A lightweight proxy that is distributed as sidecars.
- C: lets you associate multiple resources with a single domain name



POP QUIZ:

KUBERNETES DEVOPS



What is service mesh?

A: a dedicated infrastructure layer for handling service-to-service communication.

B: A lightweight proxy that is distributed as sidecars.

C: lets you associate multiple resources with a single domain name

POP QUIZ:

KUBERNETES DEVOPS



What two fields configure a retry?

- A: Attempts & perTryTimeout
- B: Rate Limiter & Load shedders
- C: NameSpace & ReplicaSet



POP QUIZ:

KUBERNETES DEVOPS



What two fields configure a retry?

- A: Attempts & perTryTimeout
- B: Rate Limiter & Load shedders
- C: NameSpace & ReplicaSet

POP QUIZ:

KUBERNETES DEVOPS



What two fields configure a retry?

- A: Attempts & perTryTimeout
- B: Rate Limiter & Load shedders
- C: NameSpace & ReplicaSet



Experiment – Istio and k3d



Tracing

winning



What is DistributedTracing

Analyzing performance in microservices and complex systems



What's in a name?

- Distributed context propagation
- Distributed transaction monitoring
- Root cause analysis
- Service dependency analysis
- Performance / latency optimization



Implementing Tracing at the Code Level

When a **microservice receives a new request**, inject information to **continue an active trace**, or **start a new trace** spanning the end-to-end use case

Store the information in a local application model, library structure or object

When making **outbound calls** to another service, **retrieve the trace information** from the local application model and **inject spanned trace** into the call stream

Trace Spans

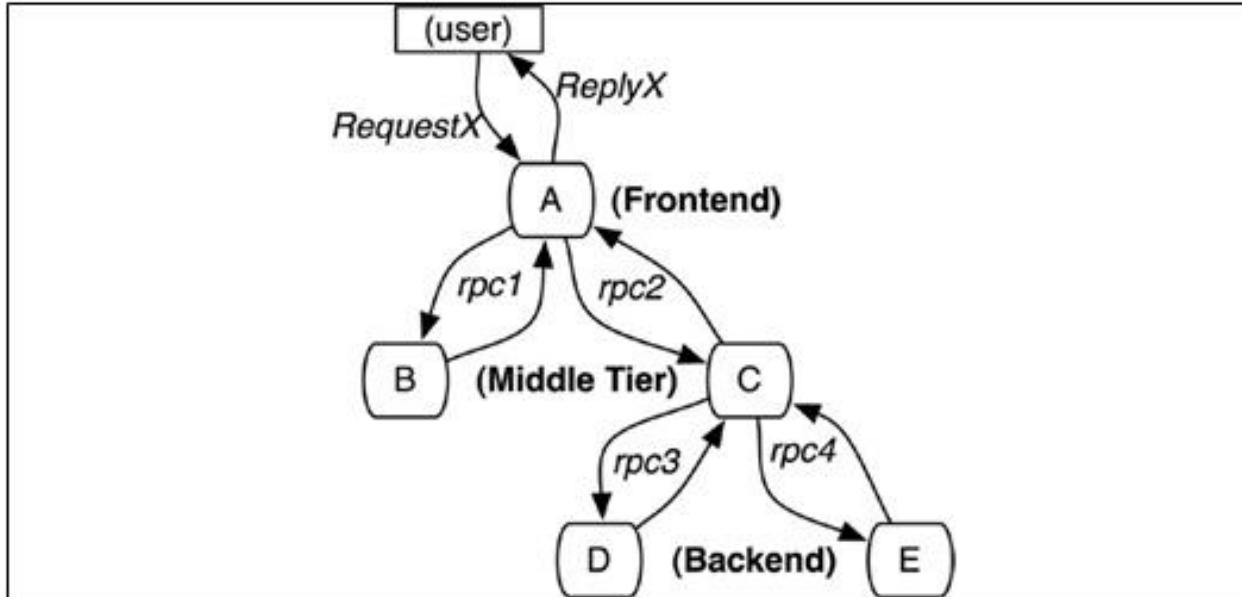


Figure 13.1: A Directed Acyclic Graph (DAG) |

Tracing Information

Traces commonly provides the following information: trace entry timestamp, parameterization, conversation identification, execution detail, and thread id

From the provided information, it is possible to understand the **program flow and execution duration** as well as the **routing across application microservices** with information.

Application Performance Monitoring

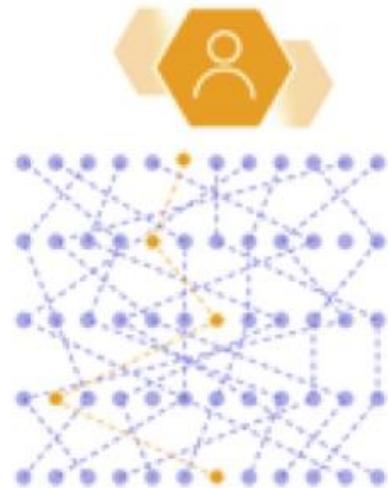
A **single standard** with the original goal of :

- Be **backwards compatible** with **OpenTracing** and **OpenCensus** instrumentation via shims.
- **Replicate the features** of both open source tracing projects before adding new



Digital Performance Monitoring

The ultimate aim of Digital Performance Monitoring is to **deliver the smoothest, fastest digital experience** to all users – whether they are **customers of a digital business** or the **workforce of a multinational enterprise**.

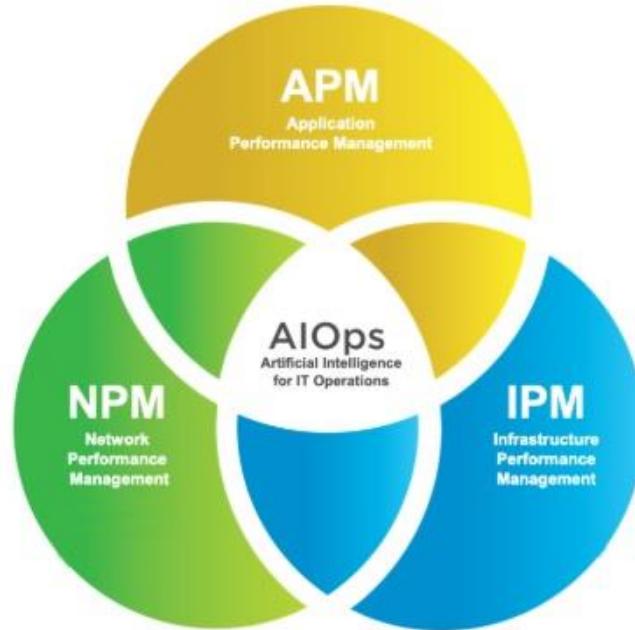


DPM Correlation

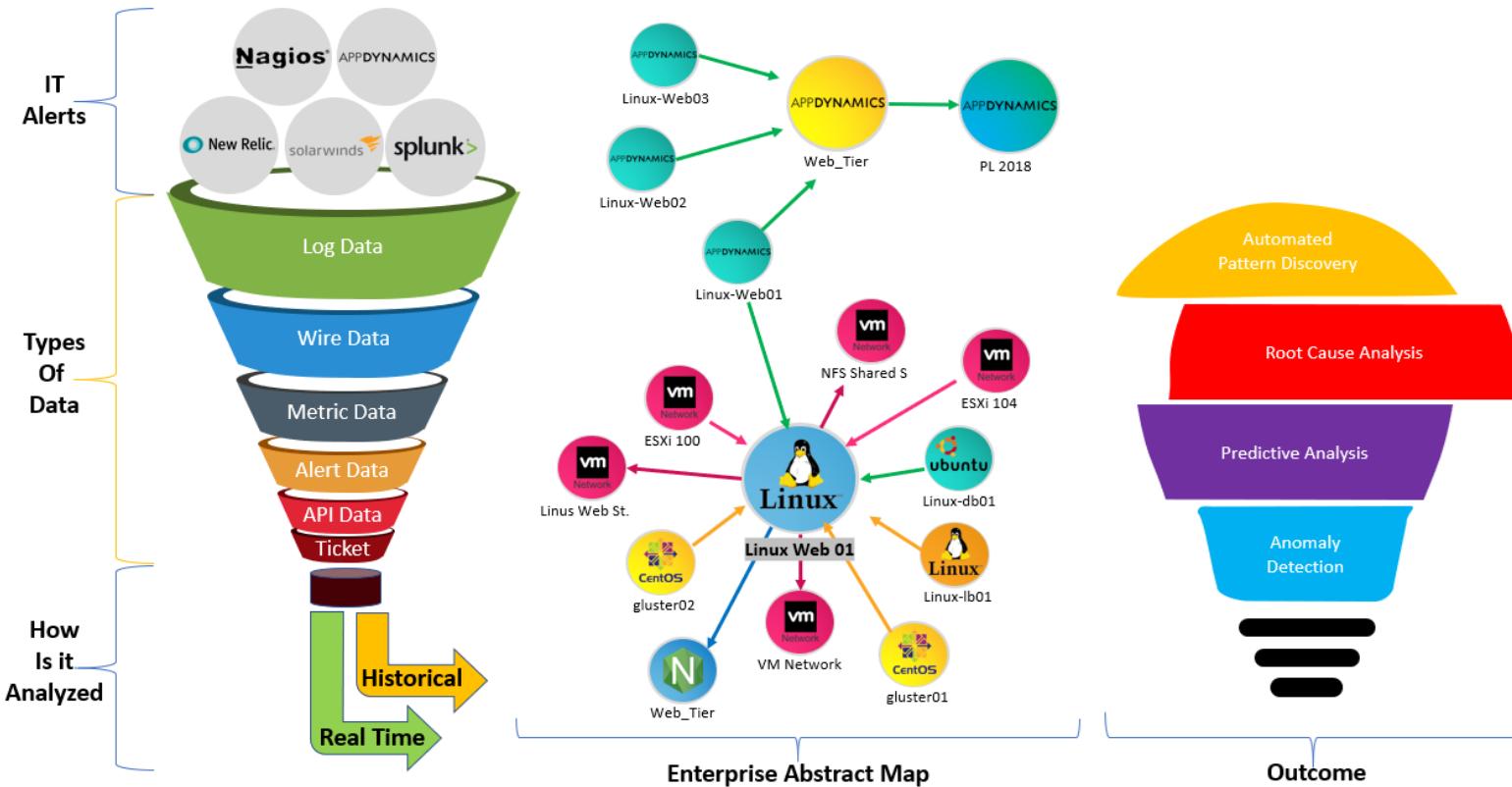


AI Ops

- Use of AIOps for event correlation and analysis, anomaly detection, root cause analysis and natural language processing
- AIOps is largely focused on pay-to-play products, e.g. Splunk, DynaTrace, AppDynamics
- AIOps focuses on domain-agnostic or domain-centric solutions
- AIOps requires data science

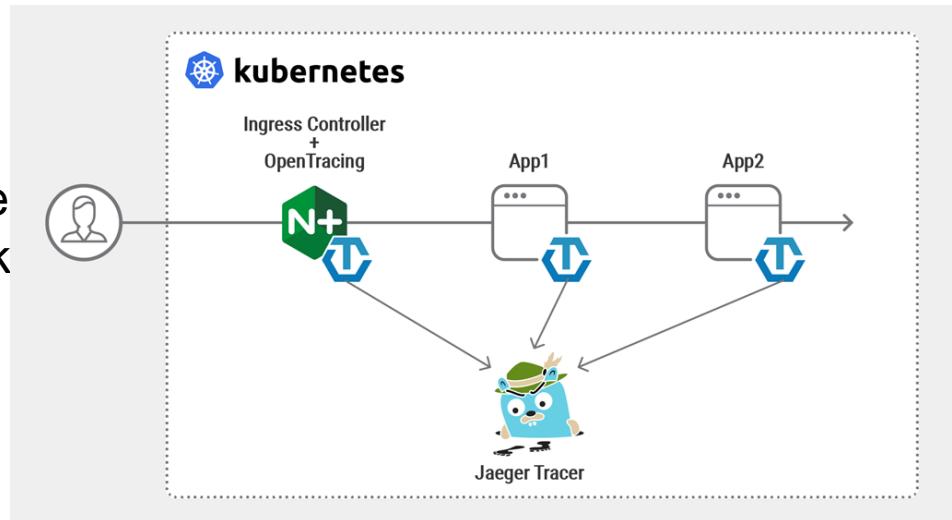


AI Ops Infographic



Manual Code Tracing

A code trace is a method for hand simulating the execution of your code in order to manually verify that it works correctly before you compile it.



OpenTracing Concepts

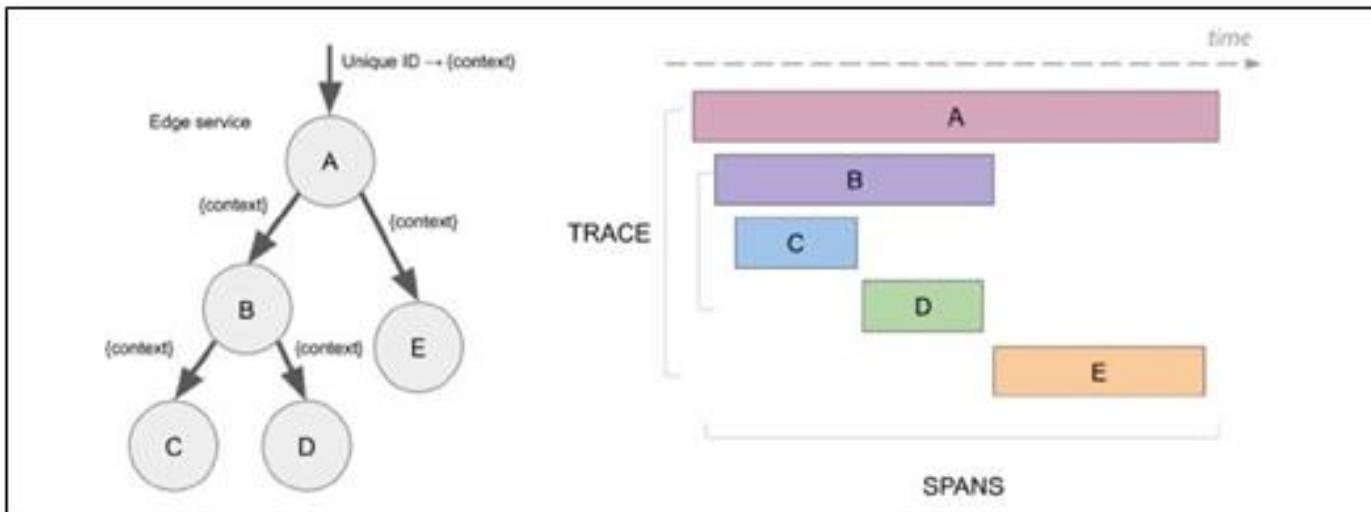
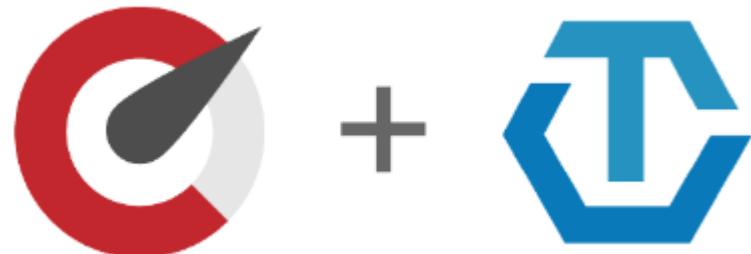


Figure 13.12: Trace and Spans relationship

Tracing Patterns

- **Agent** is a sidecar / host agent that receives telemetry from the client library in a standardized format and forwards it to collector;
- **Collector** translates the data into the format understood by a specific tracing backend and sends it there.



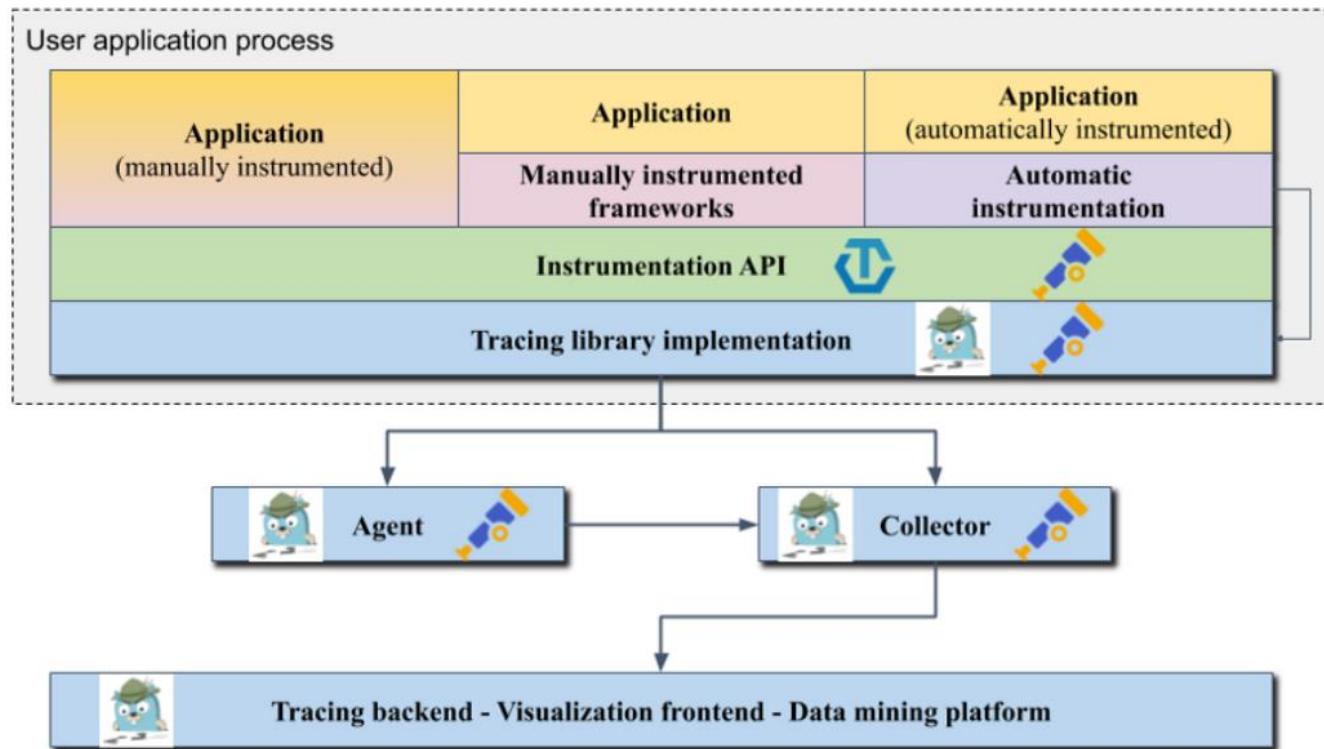
Open Telemetry

A **single standard** with the original goal of :

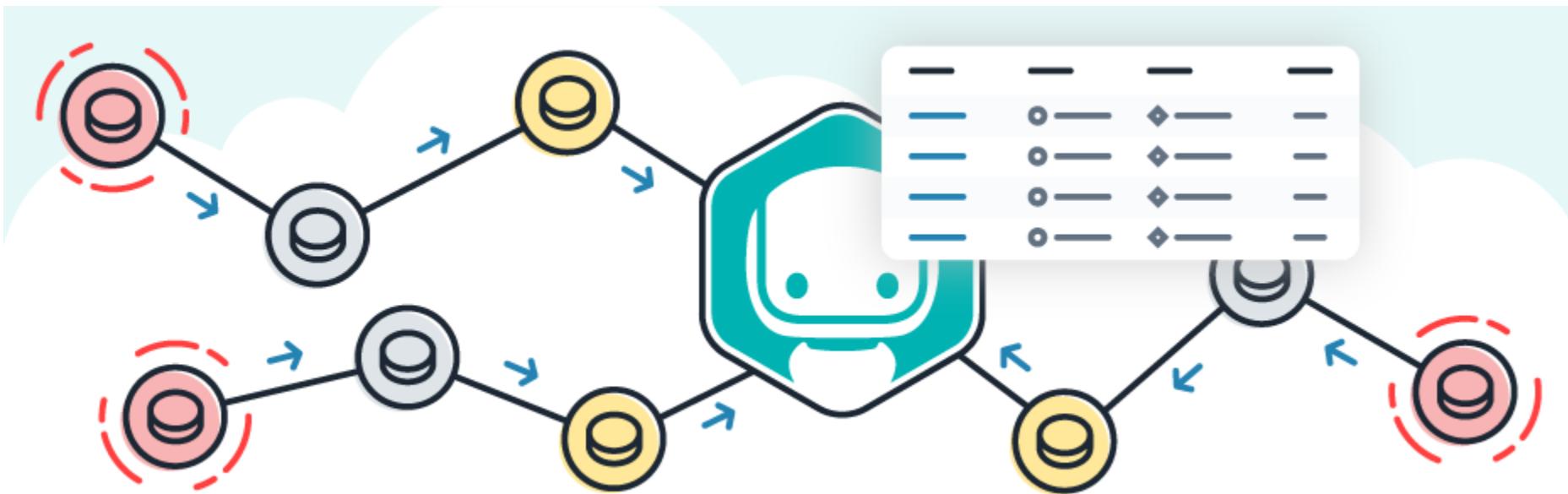
- Be **backwards compatible** with **OpenTracing** and **OpenCensus** instrumentation via shims.
- **Replicate the features** of both open source tracing projects before adding new



Open Telemetry with Jaeger



Tracing Continuity



Trace Storage Backends

Key components of observability in a cloud-native system or application include:

- NOSQL database
- Solr indexes
- Message Queuing
- In-memory storage

Collecting Telemetry from Mesh

Implementing Tracing at the Service Mesh Level

POP QUIZ:

KUBERNETES DEVOPS



What do you use Api Ops for?

A: event correlation and analysis, anomaly detection, root cause analysis and natural language processing

B: largely focused on architecture for pay-to-play products, e.g. Splunk, DynaTrace, AppDynamics

C: translates the data into the format understood by a specific tracing backend and sends it there.

POP QUIZ:

KUBERNETES DEVOPS



What do you use Api Ops for?

A: event correlation and analysis,
anomaly detection, root cause
analysis and natural language
processing

B: largely focused on architecture for
pay-to-play products, e.g. Splunk,
DynaTrace, AppDynamics

C: translates the data into the format
understood by a specific tracing
backend and sends it there.

POP QUIZ:

KUBERNETES DEVOPS



What is tracing continuity?

- A: a microservices-oriented approach to development is proving to be vital for modern SaaS business
- B: Allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.
- C: calls triggered by one external request are collected into one trace

POP QUIZ:

KUBERNETES DEVOPS



What is tracing continuity?

- A: a microservices-oriented approach to development is proving to be vital for modern SaaS business
- B: Allows developers to add instrumentation to their application code using APIs that do not lock them into any one particular product or vendor.
- C: calls triggered by one external request are collected into one trace

Jaeger Operator

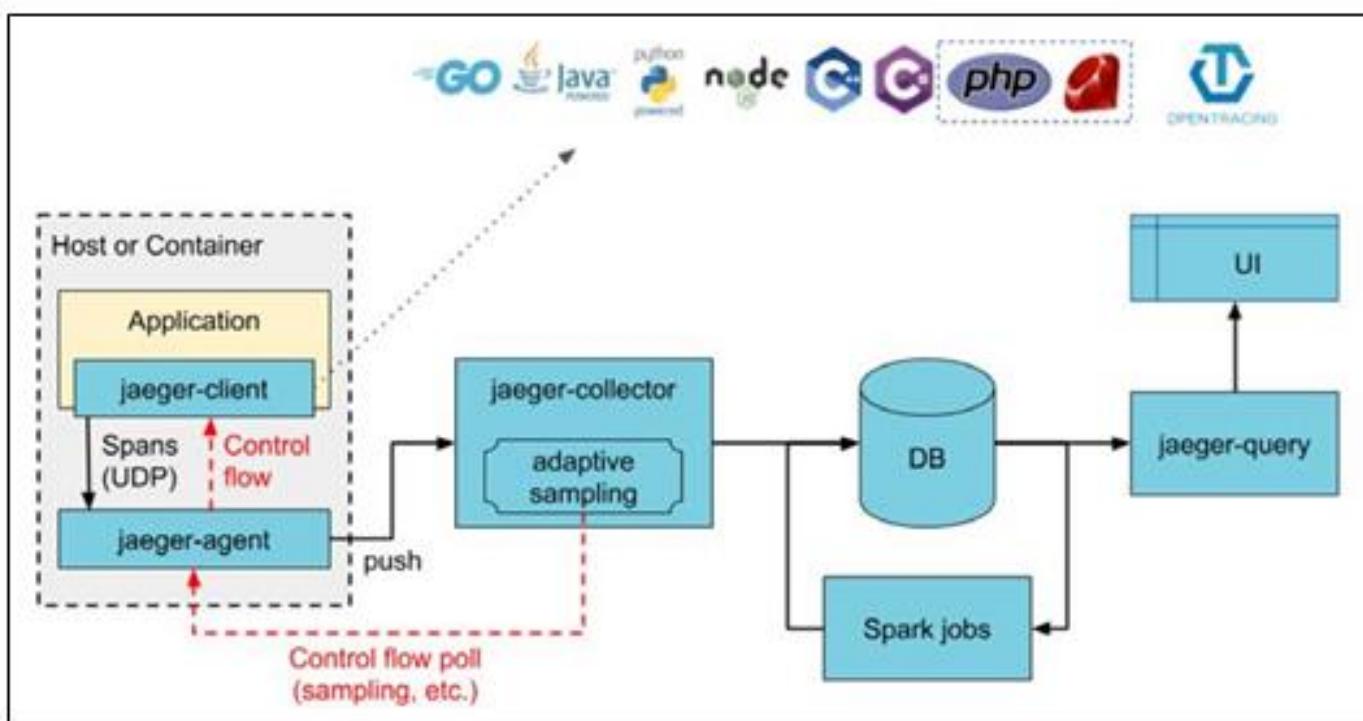


Jaeger

- Jaeger Client
- Jaeger Agent
- Jaeger Collector
- Jaeger Query



Jaeger



Installing Jaeger

```
$ helm repo add jaegertracing https://jaegertracing.github.io/helm-charts  
$ helm search repo jaegertracing
```

NAME	CHART VERSION	APP VERSION	DESCRIPTION
jaegertracing/jaeger chart for Kubernetes	0.18.3	1.16.0	A Jaeger Helm
jaegertracing/jaeger-operator Helm chart for Kubernetes	2.12.3	1.16.0	jaeger-operator

Let's install Jaeger itself first, into the monitoring namespace:

```
$ helm install jaeger jaegertracing/jaeger -n monitoring
```

NAME: jaeger

LAST DEPLOYED: Fri Jun 12 20:03:24 2020

NAMESPACE: monitoring

STATUS: deployed

REVISION: 1

TEST SUITE: None

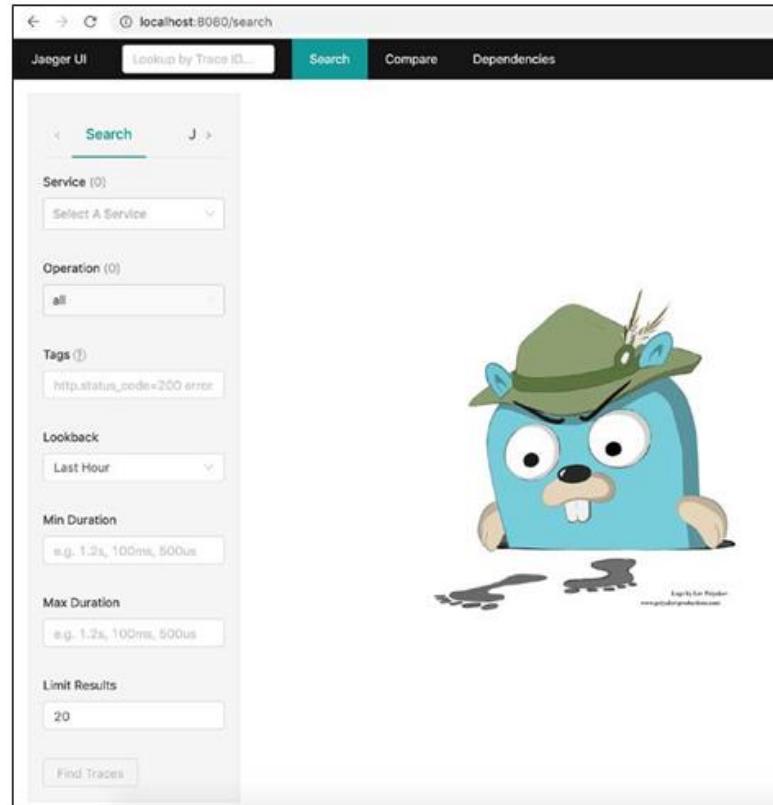
NOTES:

You can log into the Jaeger Query UI here:

```
export POD_NAME=$(kubectl get pods --namespace monitoring -l "app.jaegertracing.io/instance=jaeger,app.kubernetes.io/component=query" -o yaml | jsonpath='{.items[0].metadata.name}')  
echo http://127.0.0.1:8080/  
kubectl port-forward --namespace monitoring $POD_NAME 8080:16686
```

Jaeger Verification

Now, we can browse to
`http://localhost:8080` and see the Jaeger
UI:



The screenshot shows the Jaeger UI search interface at `localhost:8080/search`. The interface includes a search bar, dropdown menus for Service, Operation, and Tags, and input fields for Lookback, Min Duration, and Max Duration. A limit results dropdown is set to 20, and a 'Find Traces' button is present. To the right of the search form is a large, friendly blue gopher mascot wearing a green hat, standing on a small pile of mud. The gopher is smiling and has a speech bubble that reads "Especially for Developers". The URL in the address bar is `localhost:8080/search`.

Jaeger Operator

The Jaeger Operator is an implementation of a Kubernetes Operator. As we discussed previously, operators are pieces of software that ease the operational complexity of running another piece of software. More technically, *Operators* are a method of packaging, deploying, and managing a Kubernetes application.



Experiment – Jaeger Operator



POP QUIZ:

KU



What is Jaeger?





Reference

Develop a passion for
learning.

© 2019 Innovation in Software (2019)

BOOK: ANTIFRAGILE SOFTWARE

<https://leanpub.com/antifragilesoftware>

© 2019 by Innovation In Software Corporation

Written by Russ Miles with Sylvain Hellegoarch, and Grant Tarrant-Fisher.

These are the folks who have driven a boatload of the progress in Chaos Engineering and the Chaos Toolkit in the last two years.

SITE: MEDIUM.COM

<https://medium.com/search?q=kind>

<https://medium.com/search?q=k3d>

<https://medium.com/search?q=observability>

Medium.com is an emerging website with comprehensive technical content for developers. The content is detailed and with plenty of examples. Great online resource for cloud-native engineering content.

Kubernetes @ Edge

<https://forums.rancher.com/>

Rancher Labs are arguably the folks have that driven the work on Kubernetes at the Edge and IoT. Their forum is a great place to learn more about k3d and implementation for IoT and Edge computing

© 2019 by Innovation In Software Corporation