

Kubernetes Observability



Kubesimplify

Workshops



Michael Friedrich
Senior Developer Evangelist at GitLab



@dnsmichi

EveryoneCanContribute.com

Intro and preparations

Monitoring, quo vadis

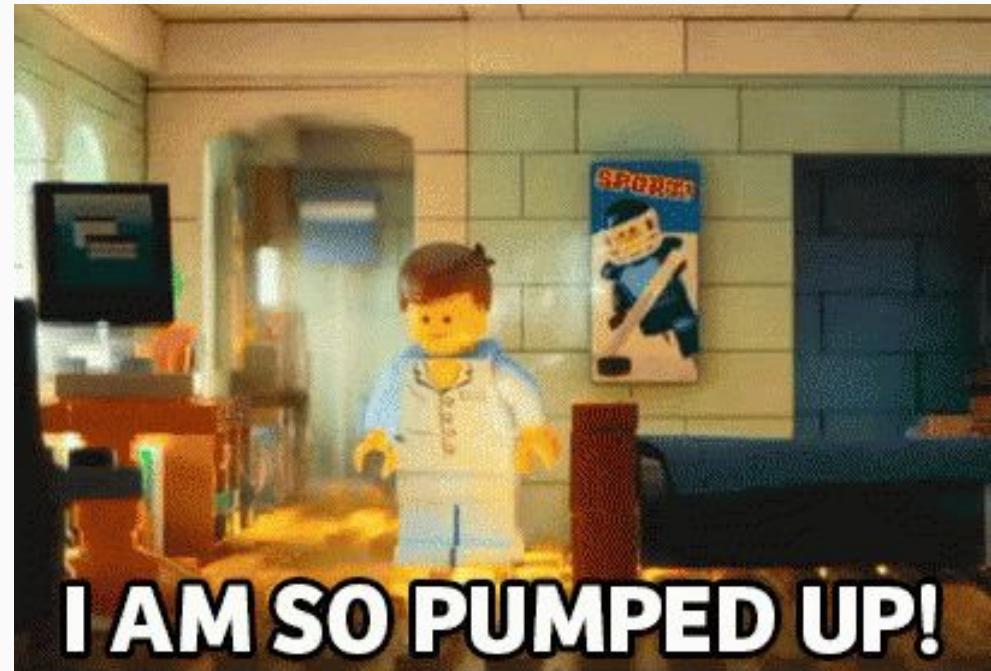
Prometheus, Grafana, Kubernetes

Alerts, incidents, SLOs

Observability?

Tracing, logs, and more

Scaling, eBPF, security, ideas



- Tools

- A fresh running Kubernetes cluster and local kubectl configured
- Admin permissions on the cluster (<https://kubernetes.io/docs/setup/>)
- Access to a hosted Git platform to persist learning state
 - Verified GitLab.com account optional

- Helpful Experience

- Ops knowledge on running and maintaining services and applications
- Dev knowledge on how to write code and define application limits
- Sec knowledge on potential vulnerabilities

```
$ <Shell commands to execute>
```

- Exercises and solutions
 - <https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-011y-2022>
- Additional projects in the groups
 - Clone instructions in the exercises
 - <https://gitlab.com/everyonecancontribute/observability>

```
$ git clone https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-011y-2022.git
```

- Local, hyper cloud or managed k*s - your choice
- Local with kind, ...
- Cloud examples
 - Amazon EKS with eksctl and kubectl:
<https://docs.aws.amazon.com/eks/latest/userguide/getting-started-eksctl.html>
 - Google GKE: <https://cloud.google.com/kubernetes-engine/docs/quickstart>
 - Civo Cloud: <https://www.civo.com/learn/kubernetes-cluster-administration-using-civo-cli>

- Install civo CLI
- Configure API token
- Create Kubernetes cluster

```
$ civo kubernetes create ks-k8s-olly  
  
$ civo kubernetes ls  
$ civo kubernetes show ks-k8s-olly  
  
$ civo kubernetes config ks-k8s-olly --save  
--merge  
  
$ kubectl config use-context ks-k8s-olly  
  
$ kubectl get node
```



Demo hosted by Civo

<https://www.civo.com/learn/kubernetes-cluster-administration-using-civo-cli>

- You can actively do the exercises
 - Or watch, and come back later to continue learning at your own pace
 - Action slides come with prefixes, EXERCISE: or Discussion
 - Some exercises are marked as “Advanced” and require more time/knowledge
- Take notes throughout the workshop
 - Local editor or online IDEs in the browser, edit README.md with Markdown
- Slides link to more resources
 - Study them async
- Monitoring/Observability with Kubernetes is huge
 - Don’t feel overwhelmed
 - Do smaller iterations, build brick by brick
 - Technology is moving fast - start with the basics
 - I don’t know everything either and will Google in case
 - Ask questions and share thoughts

Monitoring, Quo vadis



Kubesimplify

Workshops



@dnsmichi



Photo by [Michael Friedrich](#)

IT Infrastructure (server, network hardware, etc.)

Application performance

Cloud Resources and APIs

Distributed microservices

Reachable over the network

Different protocols (TCP, HTTP, SNMP, etc.)

Query a service to verify its health status

“Service Monitoring” with Host - Service relation

Ping, HTTP, <external service port> remote/local

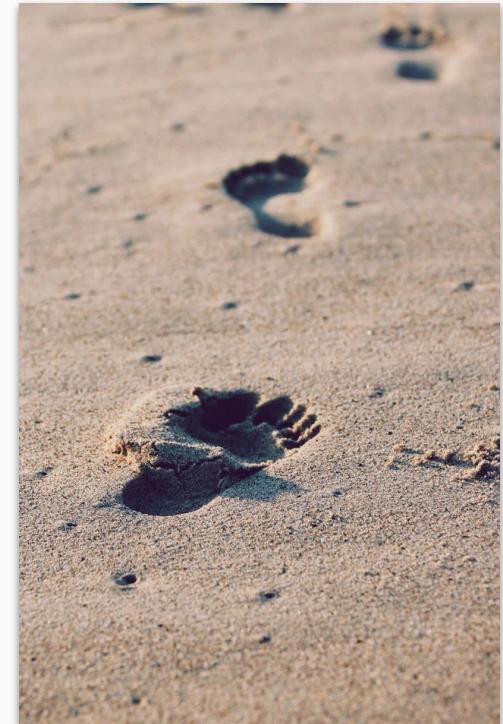
Executable “check script” to poll services

Run in interval X

Calculate state: OK, not-OK, limited text output

How to interpret the values, identify trends?

Limited to outside view. What happens inside the service?



Applications emit log lines, timestamp and context

Central log management required

- Data retention, search performance

- Monitoring integration for incident debugging

Developers learn log message formats

- Structured logging vs. raw



Photo by Christopher Sardegna on [Unsplash](#)

Classic monitoring tools: Poll services

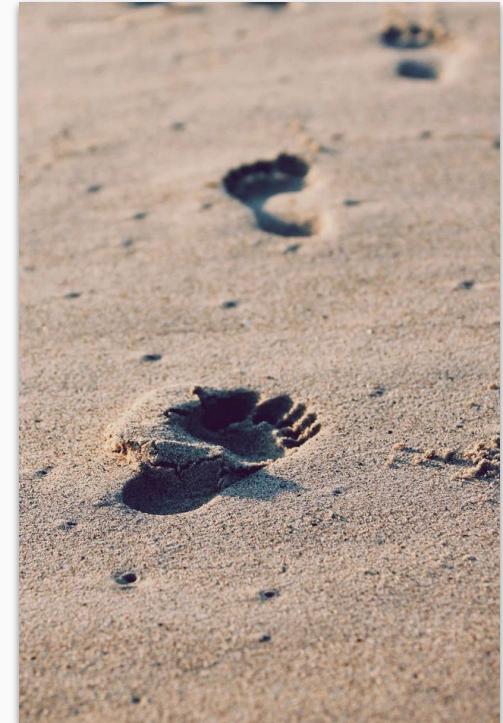
Calculate “performance data” as metric values

Monitoring tools require external storage: RRDtool,
Graphite, InfluxDB

Interval X - scheduler blocked - gaps in metric graphs

Granularity - 5min vs 10s?

Information overflow: “10TB disk is 95%”. Is that a
problem?



Host ↔ Services relationship

State based monitoring (OK, WARN, CRITICAL, UNKNOWN)

Metric values, names and tags

Volatile container names vs. static Host object names

Microservices and distributed systems need a new approach

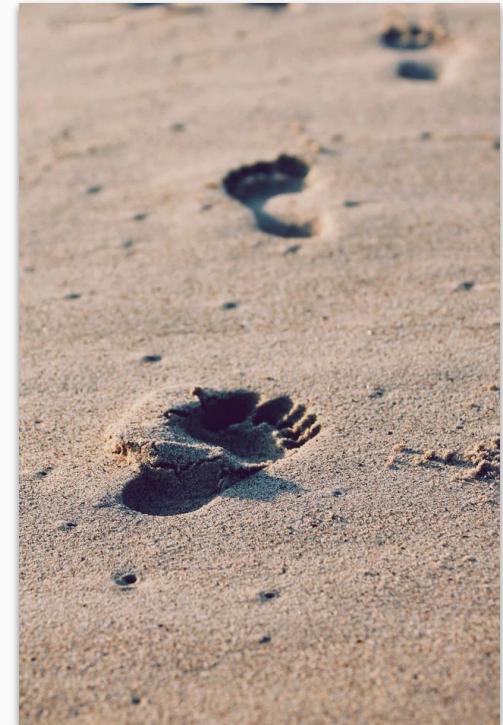


Photo by [Christopher Sardegna](#) on [Unsplash](#)

State blackbox monitoring

Traditional SLA reporting

State changes over time

Metric data points and trends

SLOly adding metrics

Whitebox monitoring

Prometheus /metrics in Docker in 2016

Add metrics output to docker #25820



thaJeztah merged 1 commit into moby:master from crosbymichael:prom  on 28 Oct 2016

<https://github.com/moby/moby/pull/25820>

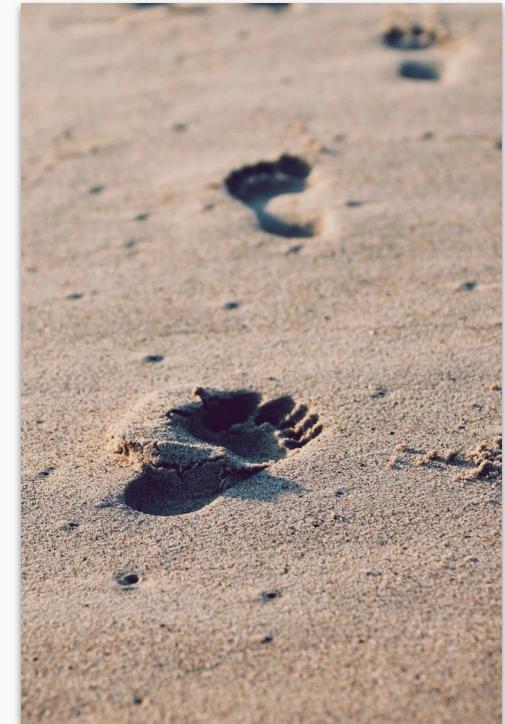


Photo by Christopher Sardegna on [Unsplash](#)

Kubernetes



Kubesimplify

Workshops



@dnsmichi



Photo by [Michael Friedrich](#)

Kubernetes

Architecture, Components

Nodes, Pods, Containers

Deployments, Services, ...

APIs, ports, data sources



We need monitoring

@dnsmichi 

Availability monitoring

Performance and resources

Identify slow or blocking deployments

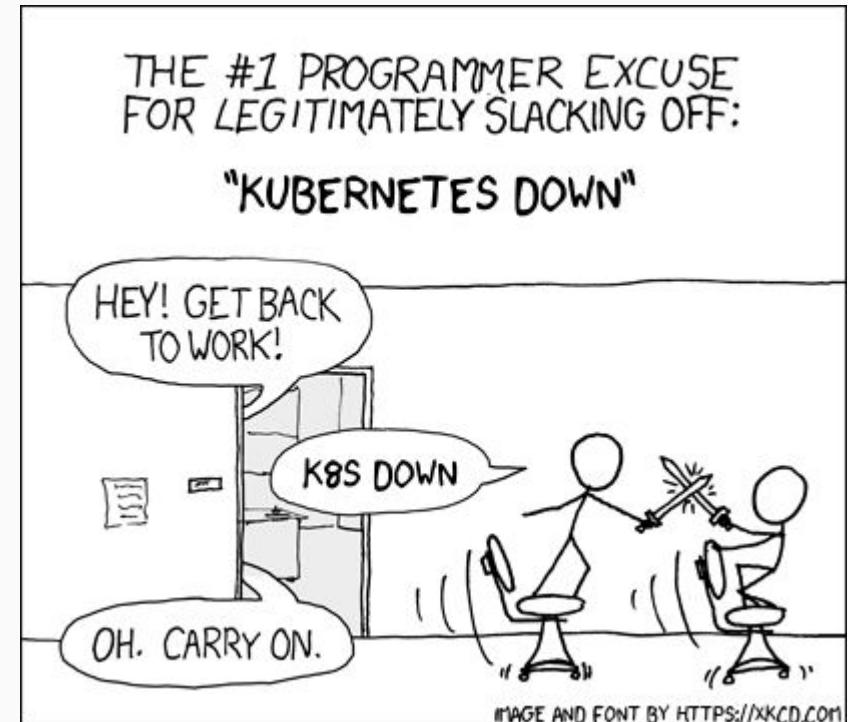
~~Host Service State based with microservices~~

Metrics? Logs? More?

Understand all components?

Best practices?

 Focus needed



- Microservices in containers run anywhere in the cluster
 - There is no hostname / IP address you can immediately poll
- Replica sets run the service on multiple pods
 - Business process models with 2 out 3 = OK don't work 1:1
- Kubernetes detects health and repairs state
 - One second it remains broken, next second it works again
 - Traditional monitoring systems may detect "flapping services" which is different behavior
- States with OK, Warning, Critical don't apply to k8s objects & clusters
 - Forget service / state based monitoring systems with Kubernetes
- Metrics and logs are highly dynamic



1. Open Monitoring, Logging and Debugging docs
2. Problems lead to debugging and asking questions
 - a. How to monitor nodes, components, services?
 - b. Resources used and consumed, bottlenecks?
 - c. Correlate cluster events

```
$ kubectl get ns  
$ kubectl get nodes  
$ kubectl get pods -A  
  
$ kubectl logs <pod>  
  
$ kubectl logs deployment/coredns -n kube-system
```

Metrics with Prometheus



Kubesimplify

Workshops



@dnsmichi

Photo by [Michael Friedrich](#)

Kubernetes

Different data sources

Service discovery?

CNCF ecosystem

Prometheus

/metrics

TSDB, trends, dashboards

SLOs

Alerts & incidents

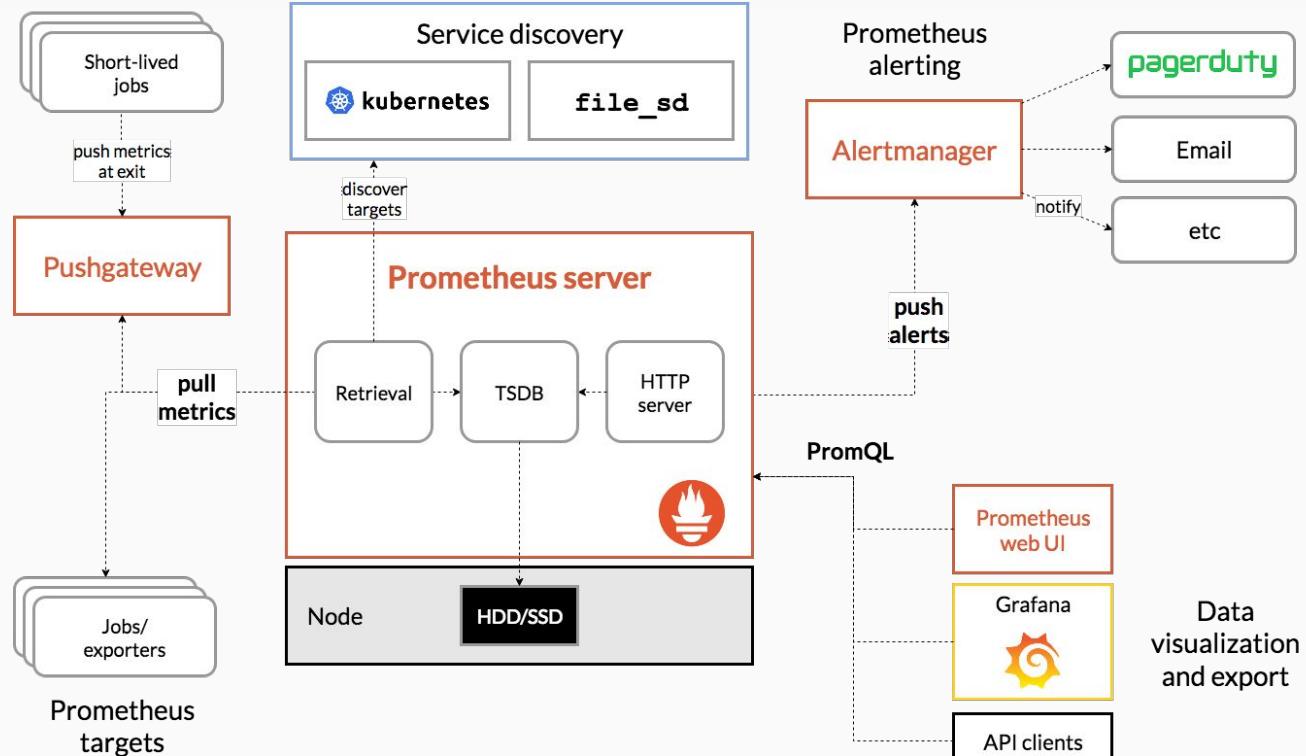


Photo by Christopher Sardegna on [Unsplash](#)

- Collect metrics from HTTP scrape targets
- Applications provide /metrics endpoint
- Rich client library ecosystem
- Service discovery for cloud native (container) environments
- Feature rich query language PromQL
- Own time-series database as backend

Prometheus: Architecture

@dnsmichi 



“Prometheus provides a functional query language called **PromQL (Prometheus Query Language)** that lets the user select and aggregate time series data in real time.

The result of an **expression** can either be shown as a **graph**, viewed as **tabular data** in Prometheus's expression browser, or **consumed** by external systems via the **HTTP API**.”

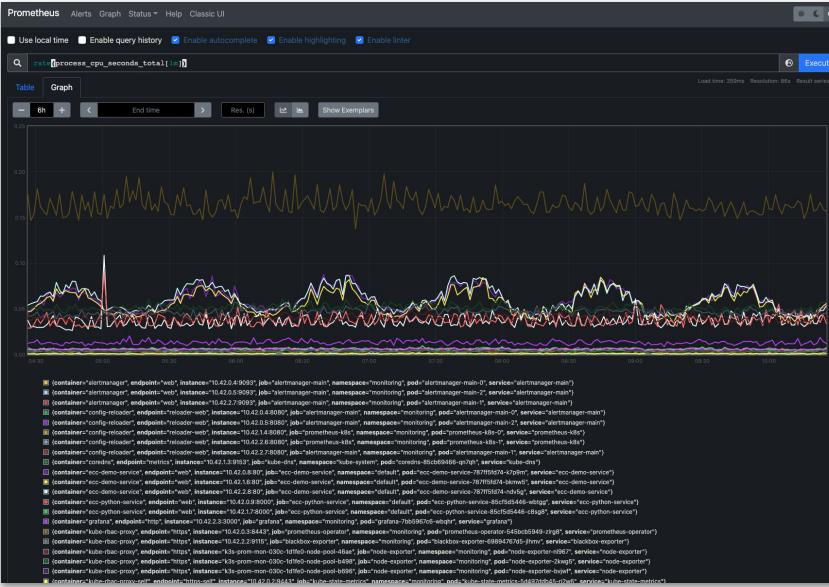
<https://prometheus.io/docs/prometheus/latest/querying/examples/>
<https://promlabs.com/promql-cheat-sheet/>
<https://demo.promlens.com/>

```
# Latest sample  
metric_name  
  
# Range  
metric_name [5m]  
  
# Labels  
metric_name{label1="a",label2="b"}  
  
# Functions  
rate(metric_name [5m])  
sum(metric_name)  
delta(metric_name [5m])  
  
# Comparisons  
metric_name > 10*1024
```

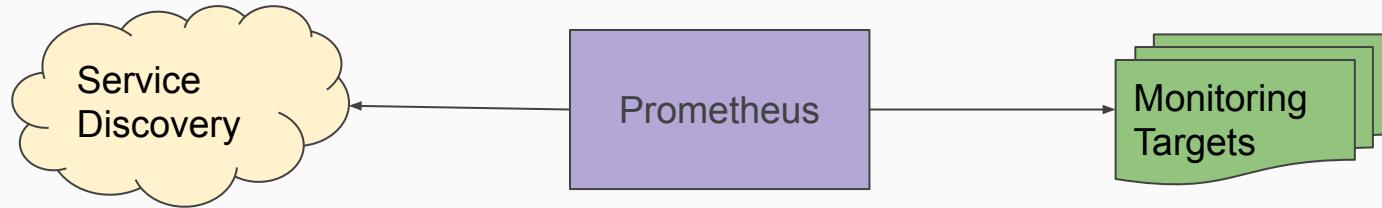
Prometheus: UI

@dnsmichi 

- UI
 - Test scrape targets
 - Query metrics
 - Graphs
- Dashboards in Grafana
 - Prometheus data source
 - PromQL query helpers
 - Panels and dashboards
- Perses
 - Dashboards as code
 - GitOps
 - WIP: <https://github.com/perses/perses>



- Lots of moving targets
 - Auto-scaling VMs in the cloud
 - Services run in containers, orchestrators shuffle workloads
 - Microservices and serverless apps
- `scrape_config`
 - Dynamic list of targets



- Kubernetes cluster goes down
 - No more monitoring?
- External monitoring?
 - No access to in-cluster resources
 - Needs sidecar agents or external APIs
 - Potential for security problems
- Deploy Prometheus server inside Kubernetes
 - Monitor its health
 - Security policies and permissions follow best practices
 - Ping probes from external, to monitor the monitor

Prometheus Operator



Kubesimplify

Workshops



@dnsmichi

The Prometheus Operator provides Kubernetes native deployment and management of Prometheus and related monitoring components. The purpose of this project is to simplify and **automate the configuration of a Prometheus based monitoring stack** for Kubernetes clusters.

The Prometheus operator includes, but is not limited to, the following features:

- **Kubernetes Custom Resources:** Use Kubernetes custom resources to deploy and manage Prometheus, Alertmanager, and related components.
- **Simplified Deployment Configuration:** Configure the fundamentals of Prometheus like versions, persistence, retention policies, and replicas from a native Kubernetes resource.
- **Prometheus Target Configuration:** Automatically generate monitoring target configurations based on familiar Kubernetes label queries; no need to learn a Prometheus specific configuration language.

kube-prometheus deploys the Prometheus Operator and already schedules a Prometheus called prometheus-k8s with alerts and rules by default. 😍

Components in a package, otherwise installed manually

Prometheus Operator, Prometheus (HA), Alert Manager (HA)

Node exporter

Prometheus Adapter for Kubernetes Metrics API, kube-state-metrics

Grafana

Pre-configured cluster monitoring

Grafana dashboards

Alerting mixins

Dynamically create ServiceDiscovery objects

<https://prometheus-operator.dev/docs/prologue/quick-start/>

<https://github.com/prometheus-operator/prometheus-operator#prometheus-operator-vs-kube-prometheus-vs-community-helm-chart>

1. <https://github.com/prometheus-operator/kube-prometheus#quickstart>
 - a. Compatibility <https://github.com/prometheus-operator/kube-prometheus#compatibility>

```
# 0. Clone prepared kube-prometheus repository (submodule ready for k8s 1.22+)

$ git clone --recursive
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-olly-2022.git && cd
k8s-olly-2022/kube-prometheus

# 1. Create monitoring namespace and custom resource definitions

$ kubectl create -f manifests/setup

# 2. Wait until the ServiceMonitor CRD is available (prints "No resources found" and exits)

$ until kubectl get servicemonitors --all-namespaces ; do date; sleep 1; echo ""; done

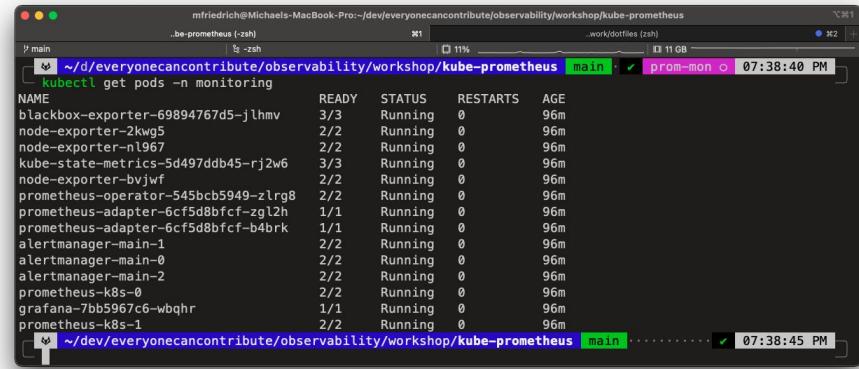
# 3. Apply remaining manifests

$ kubectl create -f manifests/
```

EXERCISE: Familiarize yourself with the deployment

@dnsmichi 

1. Inspect the monitoring namespace
2. Access to HTTP
 - a. For tests and this workshop
 - i. Add portforwarding
 - b. For production
 - i. Deploy an ingress controller
 1. Nginx, traefik, etc.



The terminal window shows the following command and its output:

```
mfriedrich@Michaels-MacBook-Pro:~/dev/everyonecancontribute/observability/workshop/kube-prometheus
$ kubectl get pods -n monitoring
NAME                      READY   STATUS    RESTARTS   AGE
blackbox-exporter-69894767d5-jlhmv   3/3     Running   0          96m
node-exporter-2kwg5                2/2     Running   0          96m
node-exporter-n1967                2/2     Running   0          96m
kube-state-metrics-5d497ddb45-rj2w6  3/3     Running   0          96m
node-exporter-bv1wf                2/2     Running   0          96m
prometheus-operator-545bcb5949-zlrg8  2/2     Running   0          96m
prometheus-adapter-6cf5d8bfcf-zg1zh  1/1     Running   0          96m
prometheus-adapter-6cf5d8bfcf-b4brk  1/1     Running   0          96m
alertmanager-main-1                2/2     Running   0          96m
alertmanager-main-0                2/2     Running   0          96m
alertmanager-main-2                2/2     Running   0          96m
prometheus-k8s-0                  2/2     Running   0          96m
grafana-7bb5967c6-wbqhr            1/1     Running   0          96m
prometheus-k8s-1                  2/2     Running   0          96m
```

```
$ kubectl get namespaces
$ kubectl get pods -n monitoring
```

```
$ kubectl --namespace monitoring port-forward
svc/prometheus-k8s 10000:9090 >/dev/null &

$ kubectl --namespace monitoring port-forward svc/grafana
20000:3000 >/dev/null &

$ kubectl --namespace monitoring port-forward
svc/alertmanager-main 30000:9093 >/dev/null &
```

1. <https://kubernetes.github.io/ingress-nginx/deploy/#quick-start>
2. Requires a domain and dynamic DNS updates

```
$ kubectl apply -f
https://raw.githubusercontent.com/kubernetes/ingress-nginx/controller-v1.1.1/deploy/static/provider/cloud/deploy.yaml

$ kubectl get pods --namespace=ingress-nginx

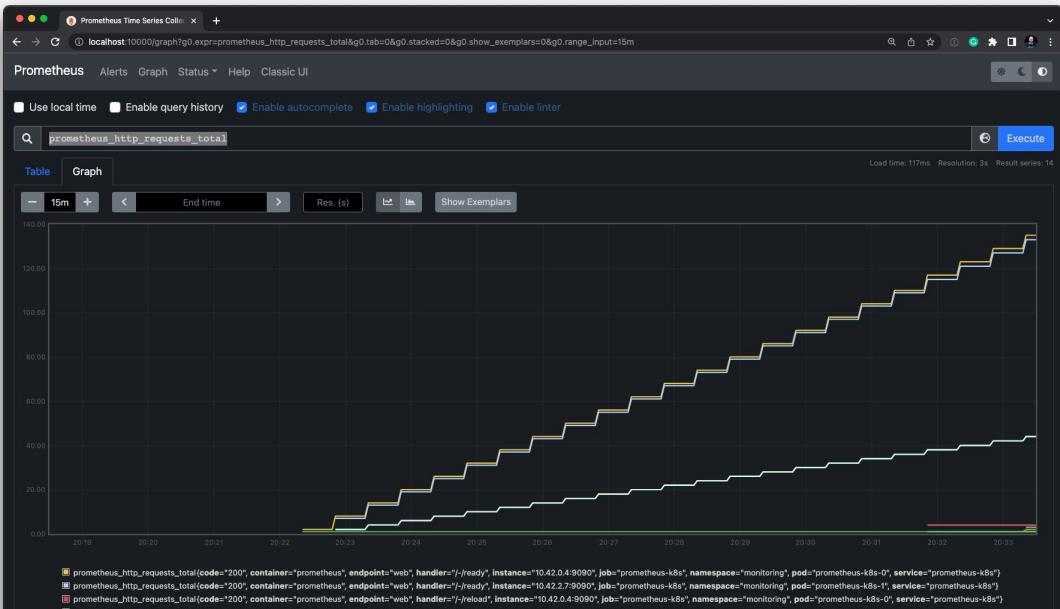
$ kubectl wait --namespace ingress-nginx \
  --for=condition=ready pod \
  --selector=app.kubernetes.io/component=controller \
  --timeout=120s

$ kubectl get service ingress-nginx-controller --namespace=ingress-nginx
```

EXERCISE: Verify Prometheus UI access

@dnsmichi 

1. Open <http://localhost:10000> to access Prometheus UI
2. Query “prometheus_http_requests_total”



```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 10000:9090 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/grafana 20000:3000 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/alertmanager-main 30000:9093 >/dev/null &
```

Kubernetes Metrics



Kubesimplify

Workshops



Photo by [Michael Friedrich](#)



@dnsmichi

Metrics?

@dnsmichi 

Cluster metrics

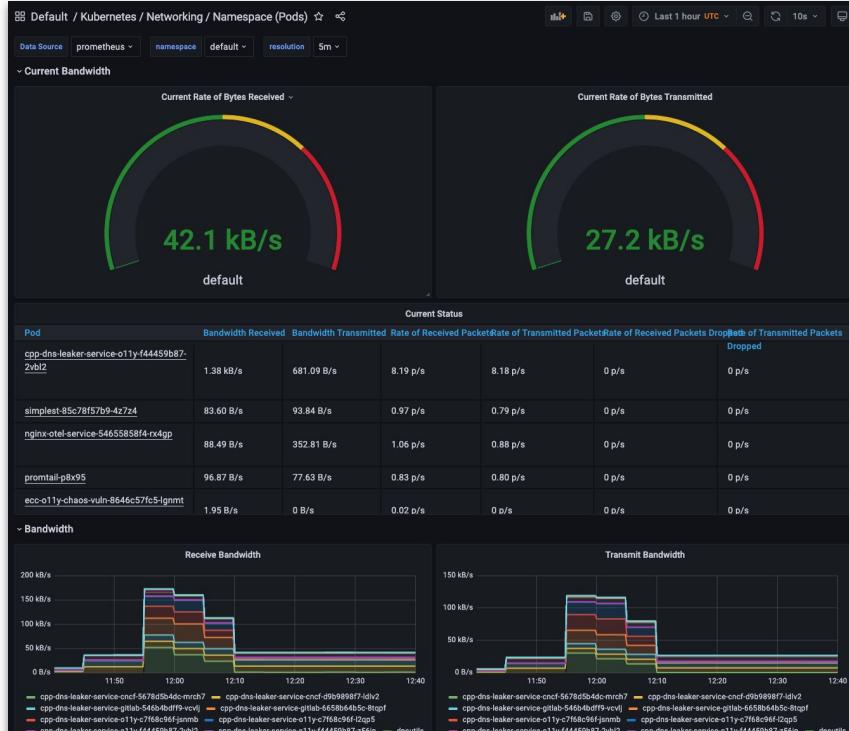
Node status, resource usage

Deployments, no of pods

Pod metrics

Kubernetes, containers, applications

So many metrics ...



Kubelets provide embedded cAdvisor exporter at /metrics/cadvisor

Resource usage monitoring, OOM kills, etc.

`container_cpu_system_seconds_total` - System CPU usage time

`container_cpu_user_seconds_total` - User CPU usage time

`container_cpu_usage_seconds_total` - Total CPU usage time
(system + user)

`container_memory_working_set_bytes` - Pod memory usage

Health of deployments, nodes and pods

Queries the Kubernetes API server

A simple service that listens to the Kubernetes API server and generates metrics about the state of the objects. It is not focused on the health of the individual Kubernetes components, but rather on the health of the various objects inside, such as deployments, nodes and pods.

Exposed metrics

Service

Deployment

Pod

Installed with the Prometheus Operator

<https://github.com/kubernetes/kube-state-metrics>

<https://github.com/kubernetes/kube-state-metrics/tree/master/docs#exposed-metrics>

Prometheus Metrics in Grafana



Kubesimplify

Workshops



Photo by [Michael Friedrich](#)



@dnsmichi

“Grafana is open source visualization and analytics software. It allows you to query, visualize, alert on, and explore your metrics no matter where they are stored. In plain English, it provides you with tools to turn your time-series database (TSDB) data into beautiful graphs and visualizations.”

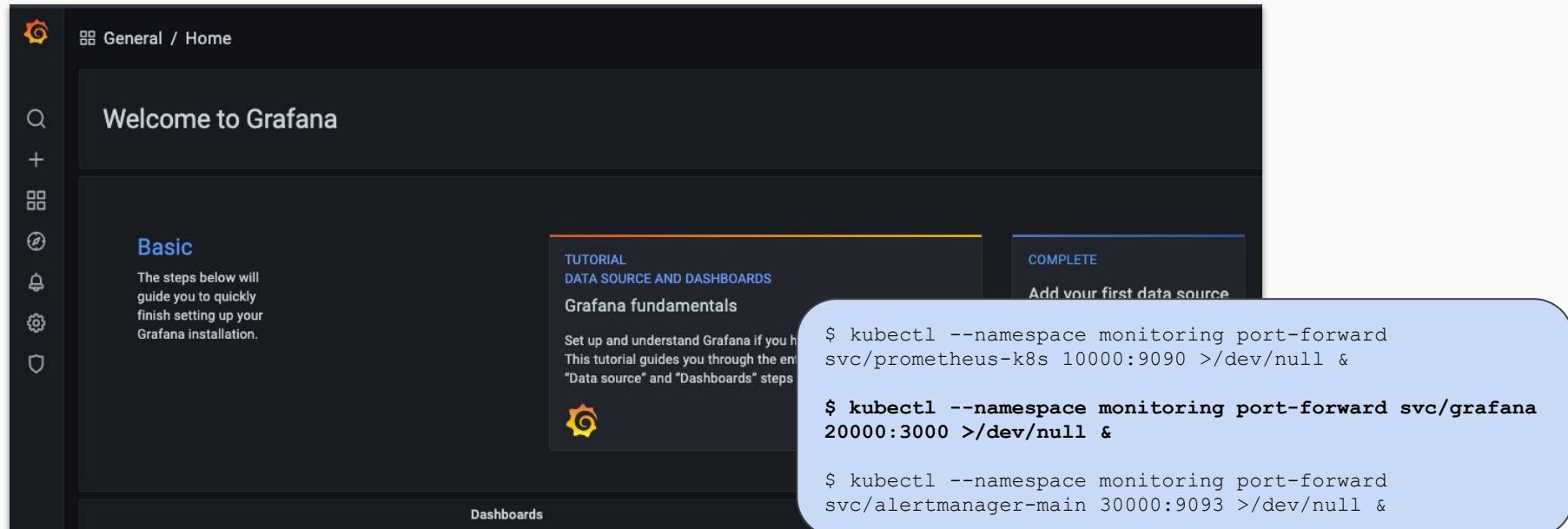
<https://grafana.com/docs/>

<https://grafana.com/docs/grafana/latest/basics/?pg=docs>

- Grafana needs external data sources
 - Graphite, InfluxDB, Prometheus
 - MySQL, Elasticsearch, ...
- Dashboards as overview
- Panels as building block with Query builder

<https://grafana.com/docs/grafana/latest/basics/glossary/?pg=docs>

1. Open <http://localhost:20000> to access Grafana UI
 - a. Grafana requires `admin:admin`, skip changing the password.



Welcome to Grafana

Basic
The steps below will guide you to quickly finish setting up your Grafana installation.

TUTORIAL
[DATA SOURCE AND DASHBOARDS](#)

Grafana fundamentals
Set up and understand Grafana if you have never used it before. This tutorial guides you through the entire process from "Data source" and "Dashboards" steps.

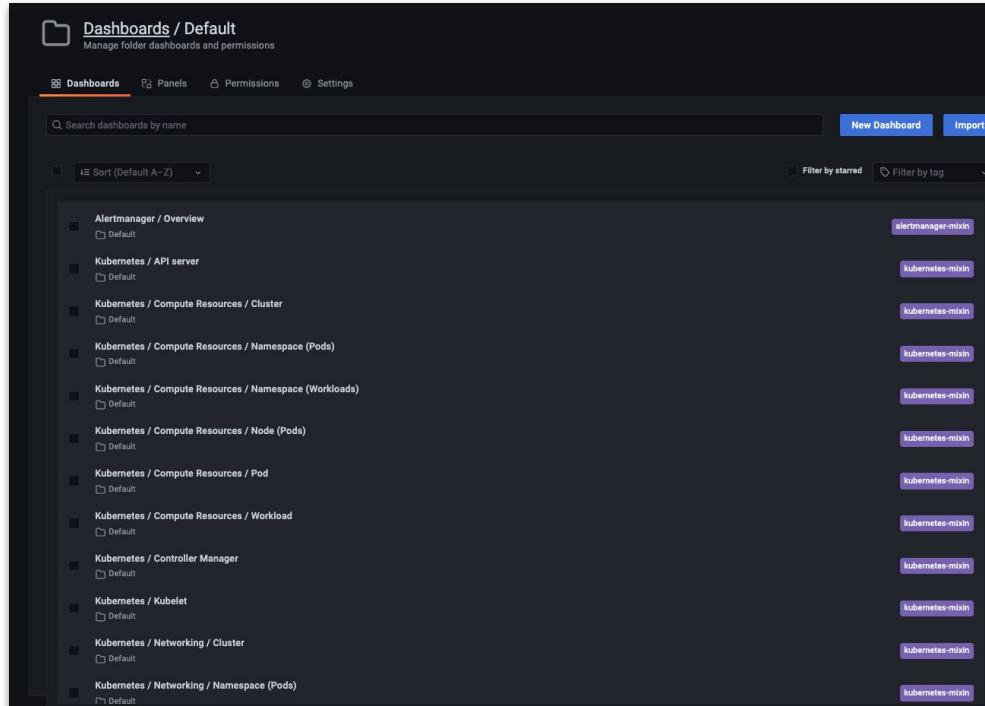
COMPLETE
[Add your first data source](#)

```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 10000:9090 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/grafana 20000:3000 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/alertmanager-main 30000:9093 >/dev/null &
```

1. Open Grafana at <http://localhost:20000>
2. Navigate to `Dashboards > Manage > Default` and inspect them
 - a. Kubernetes API Server
 - b. Compute Resources / Cluster
 - c. Kubelet
 - d. Prometheus Overview



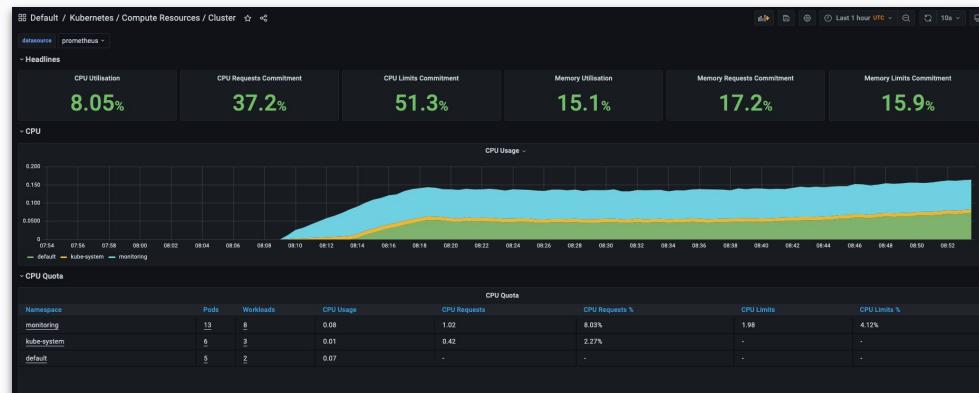
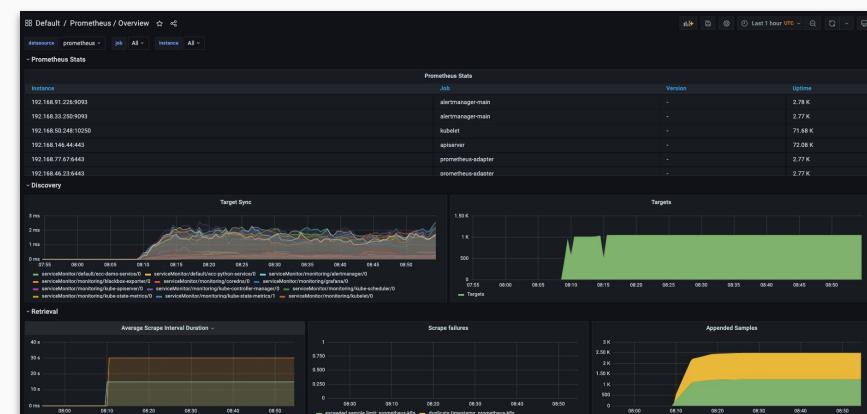
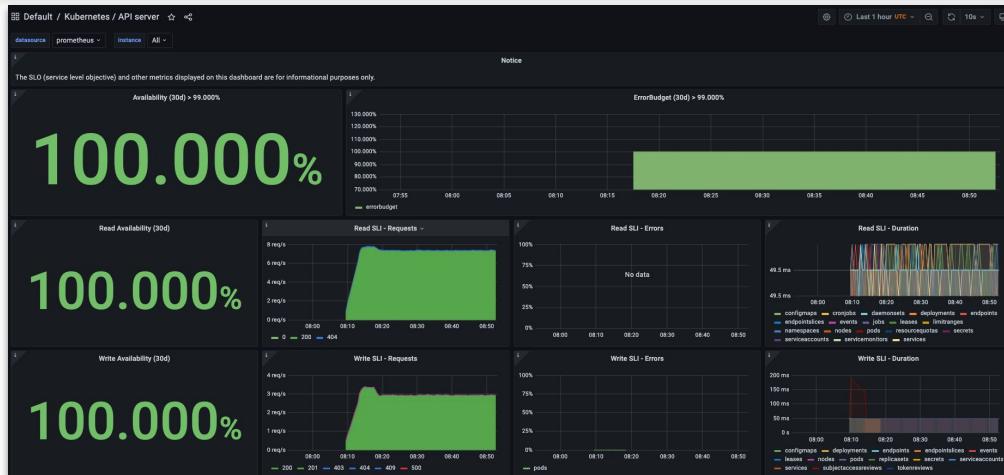
The screenshot shows the Grafana 'Dashboards / Default' management interface. The top navigation bar includes 'Dashboards' (which is selected), 'Panels', 'Permissions', and 'Settings'. Below the navigation is a search bar labeled 'Search dashboards by name' and a toolbar with 'New Dashboard' and 'Import' buttons. A dropdown menu allows sorting by name (Default A-Z) or star count. There are also 'Filter by starred' and 'Filter by tag' options.

The main content area displays a list of dashboards, each with a title, a 'Default' link, and a 'Edit' icon. To the right of each dashboard entry, there is a small purple square icon followed by the tag name 'kubernetes-mixin'. The dashboards are categorized as follows:

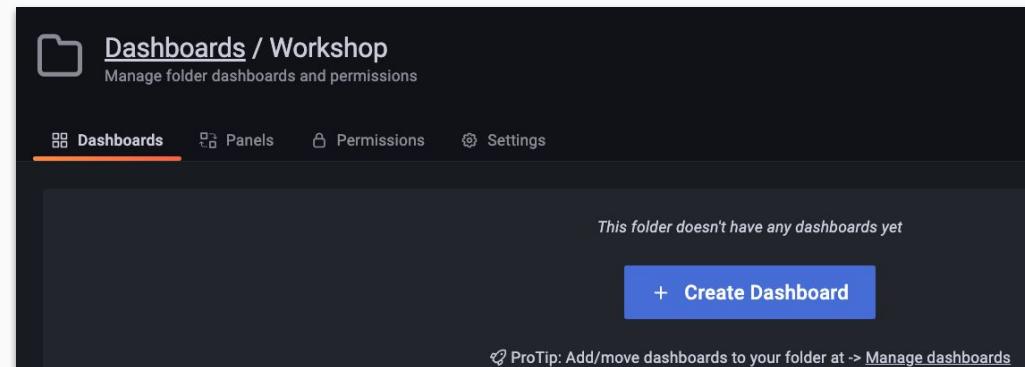
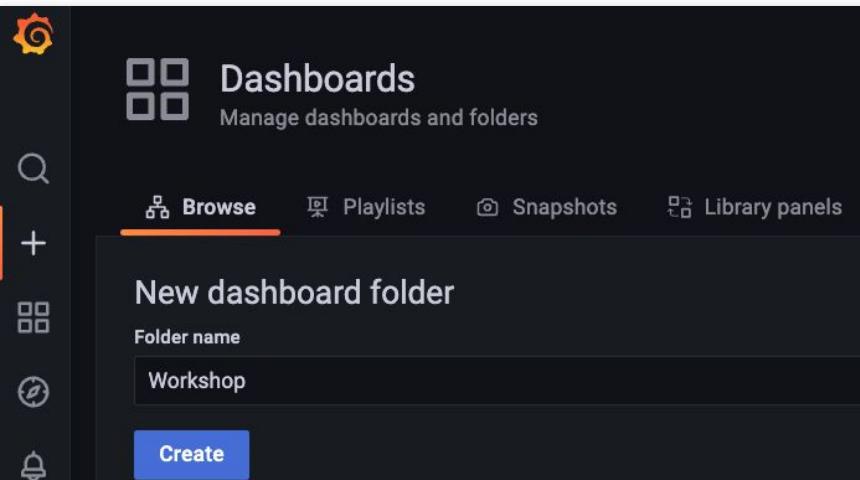
- Alertmanager / Overview
- Kubernetes / API server
- Kubernetes / Compute Resources / Cluster
- Kubernetes / Compute Resources / Namespace (Pods)
- Kubernetes / Compute Resources / Namespace (Workloads)
- Kubernetes / Compute Resources / Node (Pods)
- Kubernetes / Compute Resources / Pod
- Kubernetes / Compute Resources / Workload
- Kubernetes / Controller Manager
- Kubernetes / Kubelet
- Kubernetes / Networking / Cluster
- Kubernetes / Networking / Namespace (Pods)

Grafana dashboards

@dnsmichi 



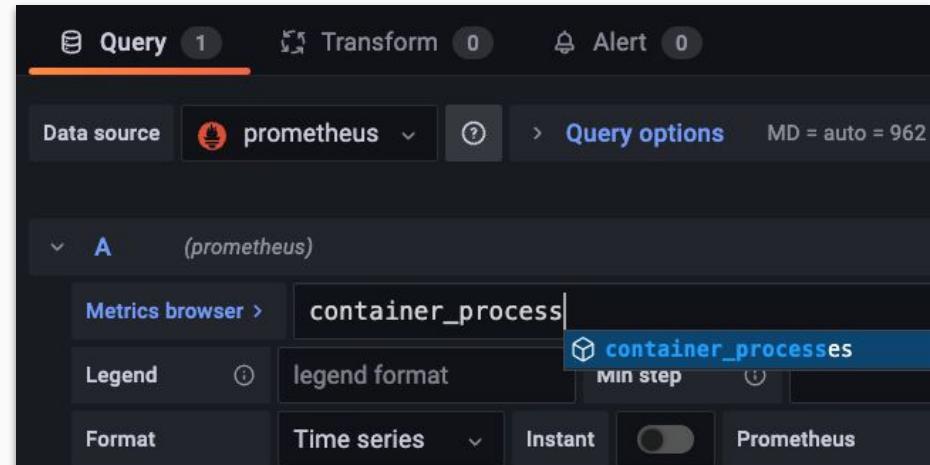
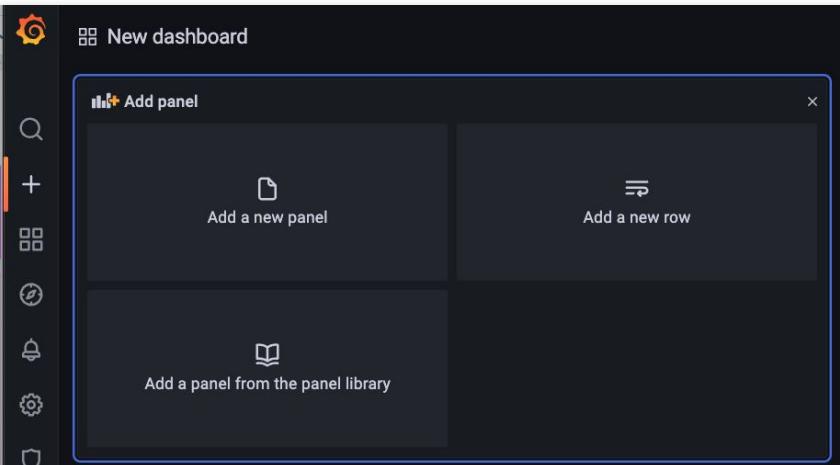
1. Open **Grafana > Dashboards > Browse at <http://localhost:20000/dashboards>**
2. Click `New Folder` and create `Workshop`, select folder & navigate inside
3. Click `+ Create Dashboard`



EXERCISE: First panel with PromQL and Grafana

@dnsmichi 

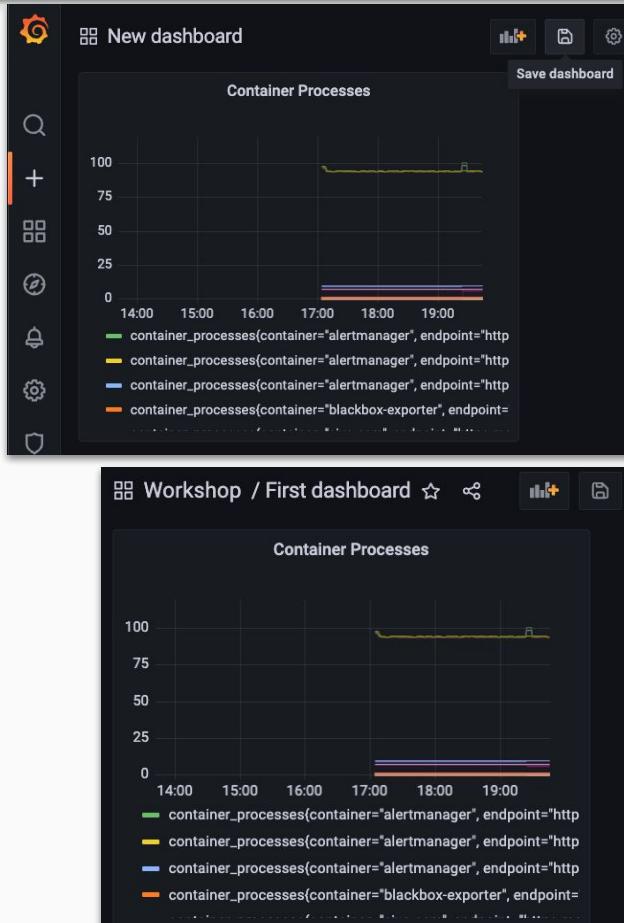
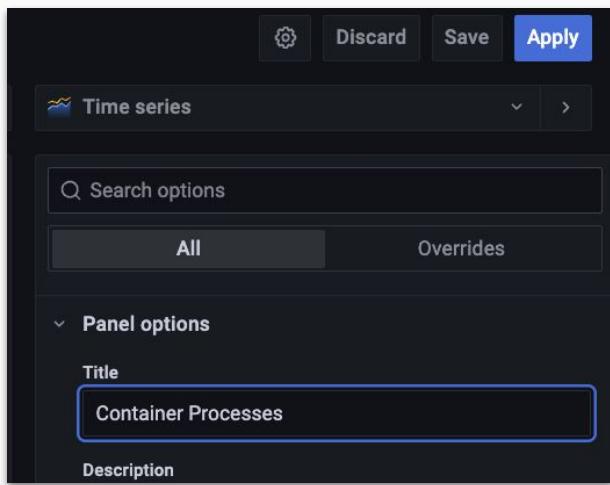
1. Create the new dashboard
2. Add a new panel
3. Data source: **Prometheus**
4. Metrics browser: **container_processes**



EXERCISE: Save the panel

@dnsmichi 

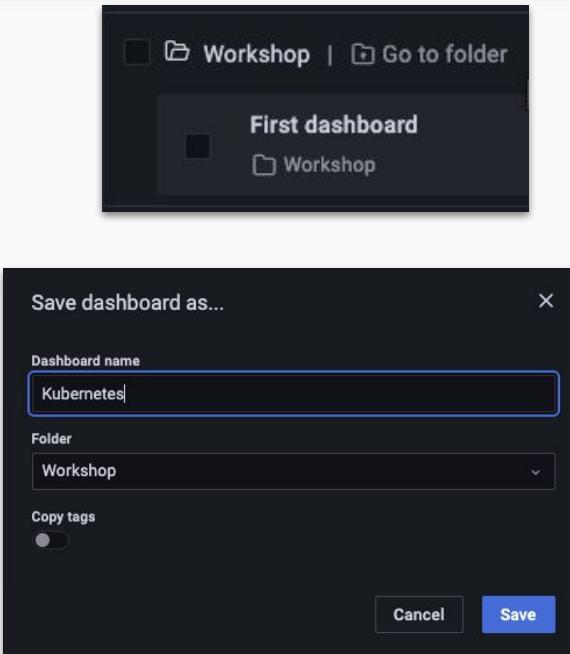
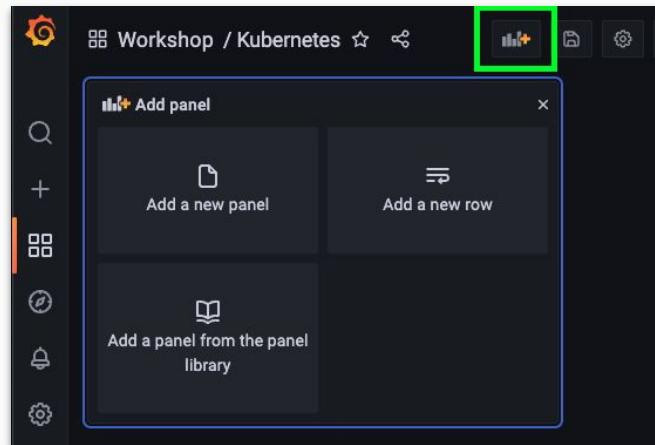
1. Edit the panel options
 - a. Title: Container Processes
 - b. Apply
2. Save the dashboard
 - a. Title: First dashboard



EXERCISE: Prepare Kubernetes Dashboard

@dnsmichi 

1. Navigate into the **Workshop** folder
2. Create new dashboard
3. Save empty
 - a. Title: Kubernetes
4. Add a new panel



A screenshot of a "Save dashboard as..." dialog box. At the top, it says "Save dashboard as...". Below that, there's a "Dashboard name" field containing "Kubernetes". Underneath it is a "Folder" dropdown menu set to "Workshop". There's also a "Copy tags" toggle switch that is turned off. At the bottom right, there are "Cancel" and "Save" buttons. The background of the dialog is dark, matching the overall theme of the Grafana interface.

- Kubelets provide embedded cAdvisor exporter:
<https://github.com/google/cadvisor> at `/metrics/cadvisor`
 - <https://kubernetes.io/docs/concepts/cluster-administration/system-metrics/#metrics-in-kubernetes>
- Resource usage monitoring, OOM kills, etc.
 - container_cpu_system_seconds_total - System CPU usage time
 - container_cpu_user_seconds_total - User CPU usage time
 - container_cpu_usage_seconds_total - Total CPU usage time (system + user)
 - container_memory_working_set_bytes - Pod memory usage

EXERCISE: Inspect Container Metrics examples

@dnsmichi 

1. Dashboards: Default > Kubernetes > Compute Resources > Cluster



1. Open the **Workshop > Kubernetes** dashboard and add a new panel
2. Query builder
 - a. Datasource: Prometheus
 - b. A: Container CPU usage rate
 - c. B (+Query): sum by namespaces
3. Save panel and add a new one
 - a. A: Memory usage by pod
 - b. B (+Query): Memory usage by pod and container
4. Save the panel and dashboard

```
rate(container_cpu_usage_seconds_total[1m])
sum by(namespace) (rate(container_cpu_usage_seconds_total[1m]))  
  
container_memory_usage_bytes{pod=~"ecc-demo-service-.*"}
container_memory_usage_bytes{pod=~"ecc-demo-service.*", container="ecc-demo-service"}
```

Health of deployments, nodes and pods

Queries the Kubernetes API server

A simple service that listens to the Kubernetes API server and generates metrics about the state of the objects. It is not focused on the health of the individual Kubernetes components, but rather on the health of the various objects inside, such as deployments, nodes and pods.

Exposed metrics

Service

Deployment

Pod

Installed with the Prometheus Operator

<https://github.com/kubernetes/kube-state-metrics>

<https://github.com/kubernetes/kube-state-metrics/tree/master/docs#exposed-metrics>

1. Open the **Workshop > Kubernetes** dashboard in Grafana
2. Add a new panel
3. Query builder
 - a. Datasource: Prometheus
 - b. Replicas
4. Title: Replica Status
5. Save the panel

```
kube_deployment_spec_replicas{job="kube-state-metrics"}
```

```
kube_deployment_status_replicas_available{job="kube-state-metrics"}
```

Prometheus server scrapes HTTP endpoints with /metrics

- Prometheus Exporters (Node, Blackbox, etc.)

- Instrumented apps

Prometheus Operator provides ServiceMonitor CRD

- Define target Endpoints

- Auto-discover targets from different namespaces

TL;DR: Provide ServiceMonitor CRD in your app deployments.

<https://prometheus-operator.dev/docs/operator/design/#servicemonitor>

EXERCISE: Deploy the demo app prometheus_demo_service

@dnsmichi 

- Create Deployment and Service resources
- Inspect and apply ecc-demo-service.yml with kubectl

```
$ kubectl apply -f  
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/prometheus-demo-service/ecc-demo-service.yml
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/prometheus-demo-service/ecc-demo-service.yml>

EXERCISE: Verify demo app prometheus_demo_service

@dnsmichi 

- Verify the created pods (3)

```
$ kubectl get pods
```



```
~ /d/e/observability/workshop/kube-prometheus main ..... ✓ | = prom-mon o 08:59:29 PM
└─ kubectl get pods
NAME          READY   STATUS    RESTARTS   AGE
ecc-demo-service-787ff5fd74-bkmw5  1/1     Running   0          88s
ecc-demo-service-787ff5fd74-ndv5g  1/1     Running   0          88s
ecc-demo-service-787ff5fd74-k7p9m  1/1     Running   0          88s
~ /dev/everyonecancontribute/observability/workshop/kube-prometheus main ..... ✓ | = 09:00:57 PM
```

The screenshot shows a terminal window with two command-line sessions. The first session, at the top, is in a directory named 'kube-prometheus' and lists three pods: 'ecc-demo-service-787ff5fd74-bkmw5', 'ecc-demo-service-787ff5fd74-ndv5g', and 'ecc-demo-service-787ff5fd74-k7p9m'. All three pods are in a 'Running' state with a 'READY' value of 1/1 and 0 restarts, having been created 88 seconds ago. The second session, at the bottom, is in a directory named 'everyonecancontribute/observability/workshop/kube-prometheus' and shows the same three pods in a similar state. The terminal interface includes a status bar at the bottom right indicating the current time.

- Prometheus at <http://localhost:10000/targets> does not monitor the service yet
 - Service exposes /metrics on port 80, name: web
- Create a CRD: **ServiceMonitor**
- Inspect and apply `ecc-demo-service-monitor.yml` with `kubectl`

```
$ kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/prometheus-demo-service/ecc-demo-service-monitor.yml

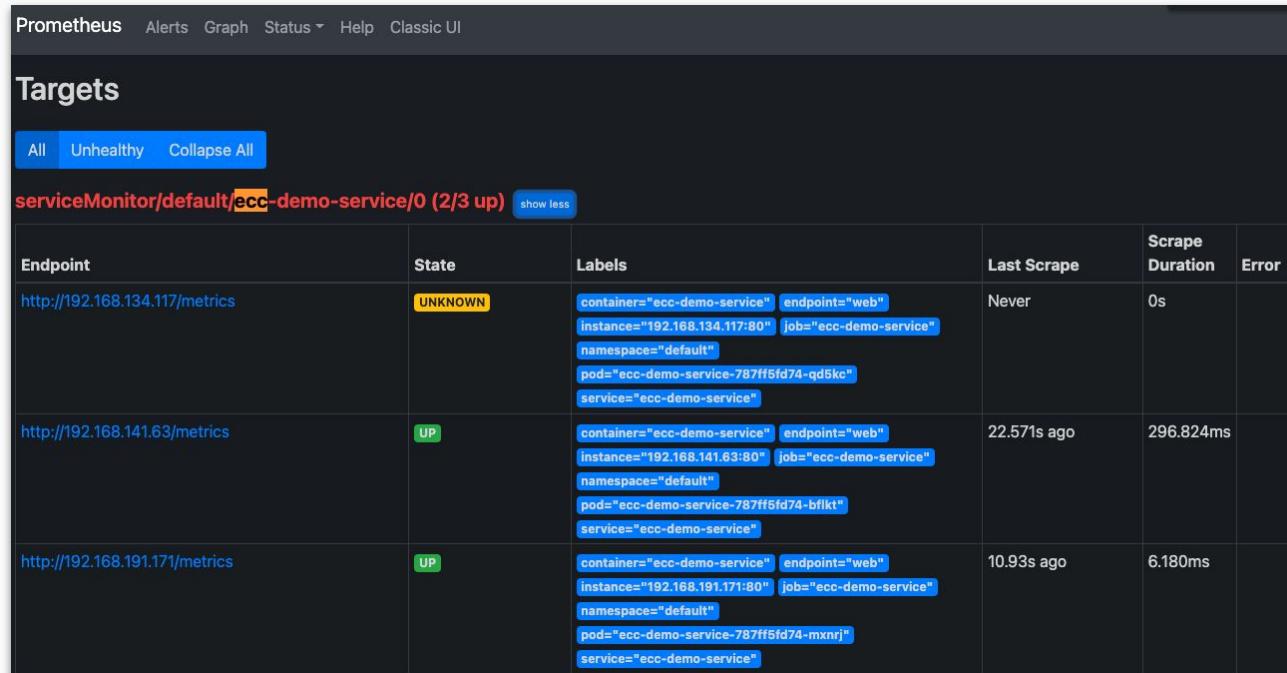
$ kubectl get servicemonitors -n monitoring
$ kubectl get servicemonitors --all-namespaces
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/prometheus-demo-service/ecc-demo-service-monitor.yml>

EXERCISE: Verify Prometheus ServiceMonitor targets

@dnsmichi 

1. Navigate to <http://localhost:10000/targets>
2. Verify the 3 **serviceMonitor** targets

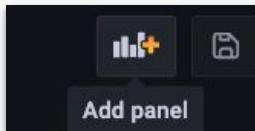


Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.134.117/metrics	UNKNOWN	<code>container="ecc-demo-service" endpoint="web" instance="192.168.134.117:80" job="ecc-demo-service" namespace="default" pod="ecc-demo-service-787ff5fd74-qd6kc" service="ecc-demo-service"</code>	Never	0s	
http://192.168.141.63/metrics	UP	<code>container="ecc-demo-service" endpoint="web" instance="192.168.141.63:80" job="ecc-demo-service" namespace="default" pod="ecc-demo-service-787ff5fd74-bflkt" service="ecc-demo-service"</code>	22.571s ago	296.824ms	
http://192.168.191.171/metrics	UP	<code>container="ecc-demo-service" endpoint="web" instance="192.168.191.171:80" job="ecc-demo-service" namespace="default" pod="ecc-demo-service-787ff5fd74-mxnri" service="ecc-demo-service"</code>	10.93s ago	6.180ms	

EXERCISE: Query demo service metrics in Grafana

@dnsmichi 

1. Add panel



2. Query

3. Data source: Prometheus

4. Metrics Browser:

a. Search for demo_cpu

b. **demo_cpu_usage_seconds_total**

c. Label: **container**

d. Values: **ecc-demo-service**

5. Use Query

```
demo_cpu_usage_seconds_total{container="ecc-demo-service"}
```

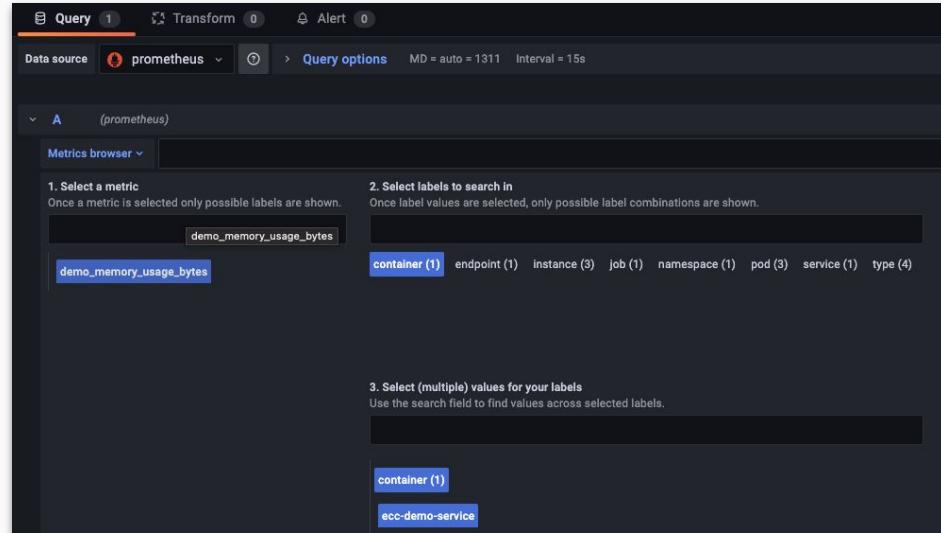
Modify the query to use the rate as suggested

```
rate(demo_cpu_usage_seconds_total{container="ecc-demo-service"}[1m])
```

The screenshot shows the Grafana Metrics Browser interface. At the top, there's a header with 'Query 1', 'Transform 0', and 'Alert 0'. Below that, it says 'Data source: prometheus' and 'Query options: MD = auto = 962 Interval = 20s'. The main area is titled '(prometheus)' and has a 'Metrics browser' dropdown. A search bar contains 'A'. Below it, there are three sections: 1. Select a metric (with a note about possible labels), 2. Select labels to search in (with a note about possible label combinations), and 3. Select (multiple) values for your labels (with a note about finding values across selected labels). The 'demo_cpu_usage_seconds_total' metric is selected in the first section. In the second section, 'container (1)' is selected. In the third section, 'ecc' is selected, and below it, 'container (1)' and 'ecc-demo-service' are listed.

The screenshot shows the Grafana Metrics Browser interface again. The search bar at the top now contains the query 'rate(demo_cpu_usage_seconds_total{container="ecc-demo-service"}[1m])'. The rest of the interface is identical to the previous screenshot, showing the 'Metrics browser' section with 'A (prometheus)' selected and various filter and value selection fields.

1. Apply the first panel
 - a. Title: CPU Usage Seconds
2. Create a new panel
3. Add empty panel > Query
 - a. Datasource: Prometheus
4. Search for **demo**
5. Build a new query
6. Use query



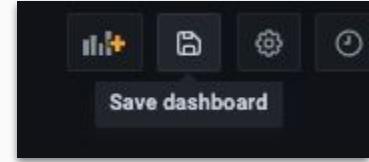
The screenshot shows the Grafana Metrics browser interface. At the top, there's a navigation bar with tabs for 'Query' (selected), 'Transform', and 'Alert'. Below the navigation is a 'Data source' dropdown set to 'prometheus' and 'Query options' with 'MD = auto = 1311' and 'Interval = 15s'. The main area is titled 'Metrics browser' and has three numbered steps:

- 1. Select a metric**: A text input field contains 'demo_memory_usage_bytes'. Below it, a list of metrics includes 'demo_memory_usage_bytes' (selected) and 'container (1)'.
- 2. Select labels to search in**: A list of labels includes 'container (1)', 'endpoint (1)', 'instance (3)', 'job (1)', 'namespace (1)', 'pod (3)', 'service (1)', and 'type (4)'.
- 3. Select (multiple) values for your labels**: A text input field contains 'container (1)'. Below it, a list includes 'ecc-demo-service'.

EXERCISE: Query demo service metrics in Grafana - practice

@dnsmichi 

1. Panel title: `CPU usage` - Apply to save
2. Save the dashboard as `Demo Service`



New dashboard / Edit Panel

Table view Fill Actual Last 6 hours Time series

CPU usage

2.20
2.00
1.80
1.60
1.40
1.20
1.00
0.800
0.600

15:15 15:30 15:45 16:00 16:15 16:30 16:45 17:00 17:15 17:30 17:45 18:00 18:15 18:30 18:45 19:00 19:15 19:30 19:45 20:00 20:15 20:30 20:45 21:00

(container=ecc-demo-service, endpoint=web, instance='192.168.134.117:80', job=ecc-demo-service, mode=idle, namespace=default, pod=ecc-demo-service-787f1f5d74-qd5kz, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.134.117:80', job=ecc-demo-service, mode=system, namespace=default, pod=ecc-demo-service-787f1f5d74-qd5kz, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.134.117:80', job=ecc-demo-service, mode=user, namespace=default, pod=ecc-demo-service-787f1f5d74-qd5kz, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.141.63:80', job=ecc-demo-service, mode=idle, namespace=default, pod=ecc-demo-service-787f1f5d74-bflkt, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.141.63:80', job=ecc-demo-service, mode=idle, namespace=default, pod=ecc-demo-service-787f1f5d74-bflkt, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.141.63:80', job=ecc-demo-service, mode=idle, namespace=default, pod=ecc-demo-service-787f1f5d74-bflkt, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.191.171:80', job=ecc-demo-service, mode=idle, namespace=default, pod=ecc-demo-service-787f1f5d74-nanji, service=ecc-demo-service)
(container=ecc-demo-service, endpoint=web, instance='192.168.191.171:80', job=ecc-demo-service, mode=system, namespace=default, pod=ecc-demo-service-787f1f5d74-nanji, service=ecc-demo-service)

Query Transform Alert Data source prometheus Query options Data source prometheus Metrics browser rate(demo_cpu_usage_seconds_total{container="ecc-demo-service"}[5m]) Resolution 1/1 Format Instant Prometheus Exemplars Legend

MD = auto = 1482 Interval = 15s Query inspector

Legend placement Bottom Right Legend values Select values or calculations to show in legend Choose

Monitoring 2.0

App instrumentation

Customizing



Kubesimplify

Workshops



@dnsmichi

SLO

Monitoring

Metrics

key/tag

Values

?



1. Learn more about Prometheus Client Libraries for Instrumentation
2. Inspect the project [prometheus_python_service](#)
 - a. app.py
3. Deploy the app and create a ServiceMonitor CRD
4. Navigate to <http://localhost:10000/targets> to verify

```
$ kubectl create -f
https://gitlab.com/everyonecancontribute/observability/prometheus\_python\_service/-/raw/main/manifests/ecc-python-service.yml

$ kubectl create -f
https://gitlab.com/everyonecancontribute/observability/prometheus\_python\_service/-/raw/main/manifests/ecc-python-service-monitor.yml
```

https://github.com/prometheus/client_python#three-step-demo

https://gitlab.com/everyonecancontribute/observability/prometheus_python_service

1. Navigate to <http://localhost:10000/targets> to verify
2. Add new Grafana dashboard: **Apps**
3. New Panel query: `rate(request_processing_seconds_count{}[1m])`

serviceMonitor/default/ecc-python-service/0 (2/2 up) show less					
Endpoint	State	Labels	Last Scrape	Scrape Duration	Error
http://192.168.133.145:8000/metrics	UP	<code>container="ecc-python-service" endpoint="web"</code> <code>instance="192.168.133.145:8000" job="ecc-python-service"</code> <code>namespace="default"</code> <code>pod="ecc-python-service-85cf5d5446-gnzhg"</code> <code>service="ecc-python-service"</code>	8.525s ago	2.873ms	
http://192.168.159.213:8000/metrics	UP	<code>container="ecc-python-service" endpoint="web"</code> <code>instance="192.168.159.213:8000" job="ecc-python-service"</code> <code>namespace="default" pod="ecc-python-service-85cf5d5446-rlfvp"</code> <code>service="ecc-python-service"</code>	19.211s ago	2.580ms	

https://github.com/prometheus/client_python#three-step-demo

https://gitlab.com/everyonecancontribute/observability/prometheus_python_service

1. Fork or clone [prometheus_python_service](#) (or watch the trainer)

- a. Modify the source code
- b. Commit & push to the main branch
- c. Wait for CI/CD to publish new Docker image
- d. Inspect Packages & Registries > Container Registry
- e. Copy the container URL into YAML config

2. Re-deploy into Kubernetes

3. Add panel in Grafana, query for `ws_`

```
$ vim manifests/ecc-python-service.yml

spec:
  containers:
    - name: ecc-python-service
      # Modify this to your fork repository registry
      image: registry.gitlab.com/everyonecancontribute/observability/prometheus_python_service:latest

$ kubectl apply -f manifests/ecc-python-service.yml

$ kubectl rollout restart deploy ecc-python-service
```



https://github.com/prometheus/client_python#three-step-demo

https://gitlab.com/everyonecancontribute/observability/prometheus_python_service

Alerts and SLOs



Kubesimplify

Workshops



Photo by [Michael Friedrich](#)

(what) Definition of failure

(why) Do something because

Reasons (e.g. threshold violated)

Rule matched: Notify and raise alerts

(who) Responsible persona/team

(how) Documentation for incidents, runbooks, corrective actions

(iterate) Mean time to response (MTTR)

<https://about.gitlab.com/handbook/engineering/infrastructure/incident-management/>

<https://gitlab.com/gitlab-com/runbooks>

Prometheus Alert Rules

Sends alerts to Alert Manager

Alert Manager

Grouping - similar nature, avoiding duplicated alerts

Inhibition - suppress alerts when others are fired already

Silences - mute alerts for a given time

Transports

SMTP (Email), Webhook, pushover, SNS

Pager (VictorOps, PagerDuty, Opsgenie)

Chat (Slack, WeChat, Telegram)

<https://prometheus.io/docs/alerting/latest/alertmanager/>
<https://prometheus.io/docs/alerting/latest/configuration/>

Infrastructure alerts

Memory, CPU, disk pressure

Pods

Error rate / response time

Reachability

```
- alert: KubernetesPodNotHealthy
  expr: min_over_time(sum by (namespace, pod)
  (kube_pod_status_phase{phase=~"Pending|Unknown|Failed"}) [15m:1m]
) > 0
  for: 0m
  labels:
    severity: critical
  annotations:
    summary: Kubernetes Pod not healthy (instance {{ $labels.instance }})
    description: "Pod has been in a non-ready state for longer than 15 minutes.\n  VALUE = {{ $value }}\n  LABELS = {{ $labels }}"

# Example from
https://awesome-prometheus-alerts.grep.to/rules.html#kubernetes
```

- PrometheusRule CRD
 - Wraps Prometheus rules format in the operator

```
spec:  
  groups:  
    - name: kube-state-metrics  
      rules:  
        - alert: KubeStateMetricsListErrors  
          annotations:  
            description: kube-state-metrics is experiencing errors at an elevated rate in list operations. This is  
likely causing it to not be able to expose metrics about Kubernetes objects correctly or at all.  
            runbook_url: https://github.com/prometheus-operator/kube-prometheus/wiki/kubestatemetricslisterrors  
            summary: kube-state-metrics is experiencing errors in list operations.  
          expr: |  
            (sum(rate(kube_state_metrics_list_total{job="kube-state-metrics",result="error"}[5m]))  
             /  
             sum(rate(kube_state_metrics_list_total{job="kube-state-metrics"}[5m])))  
             > 0.01  
          for: 15m  
          labels:  
            severity: critical
```

<https://prometheus-operator.dev/docs/operator/api/#prometheusrule>

https://prometheus.io/docs/prometheus/latest/configuration/alerting_rules/

<https://github.com/prometheus-operator/kube-prometheus/blob/7a3879b/manifests/kube-state-metrics-prometheusRule.yaml>

Chat: “See something immediately”

Refinements and grouping needed

Ticket and issue systems

Mailing lists

Avoid alert fatigue

Dedicated incident and issue management platforms

Trigger alerts manually - kill some pods

Manual work is a bug!

- namespace="monitoring" + 17 alerts

2021-06-11T19:06:26.729Z - Info  Source  Silence

description: Pod monitoring/node-exporter-27p57 has been in a non-ready state for longer than 15 minutes.

runbook_url: <https://github.com/prometheus-operator/kube-prometheus/wiki/kubepodnotready>

summary: Pod has been in a non-ready state for more than 15 minutes.

alertname="KubePodNotReady" +

pod="node-exporter-27p57" +

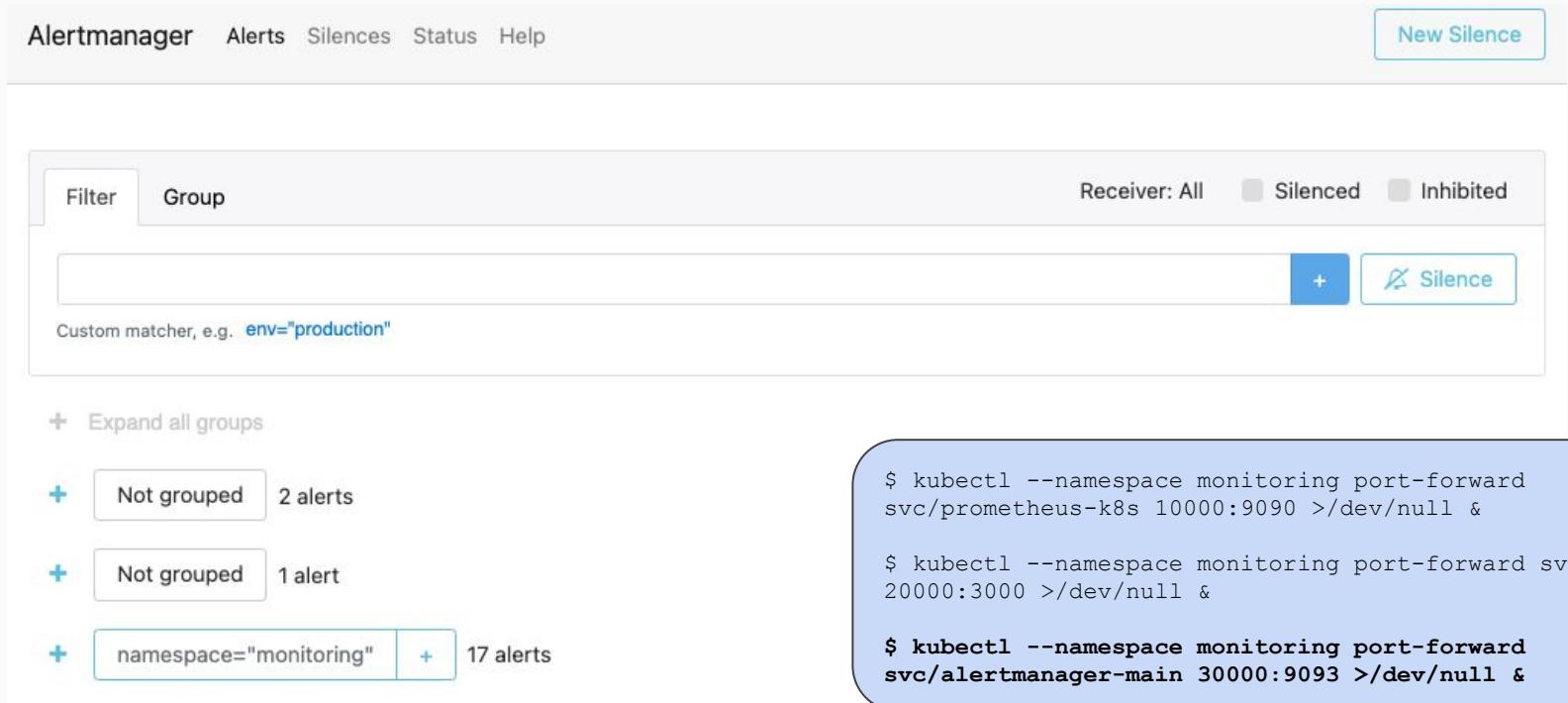
prometheus="monitoring/k8s" +

severity="warning" +

EXERCISE: Verify setup with Alert Manager UI

@dnsmichi 

1. Open <http://localhost:30000> to access Alert Manager UI



The screenshot shows the Alertmanager UI interface. At the top, there are navigation links: Alertmanager, Alerts, Silences, Status, Help, and a 'New Silence' button. Below the header is a search bar with 'Filter' and 'Group' buttons, and a 'Custom matcher, e.g. env="production"' placeholder. To the right of the search bar are buttons for 'Receiver: All', 'Silenced', and 'Inhibited'. A large blue button labeled '+' is positioned next to a 'Silence' button with a crossed-out alarm icon.

Below the search area, there are three alert categories:

- + Expand all groups
- + Not grouped 2 alerts
- + Not grouped 1 alert
- + namespace="monitoring" + 17 alerts

A callout box highlights the 'namespace="monitoring"' filter, showing the following command:

```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 10000:9090 >/dev/null &
```

Another callout box highlights the '17 alerts' count, showing the following command:

```
$ kubectl --namespace monitoring port-forward svc/grafana 20000:3000 >/dev/null &
```

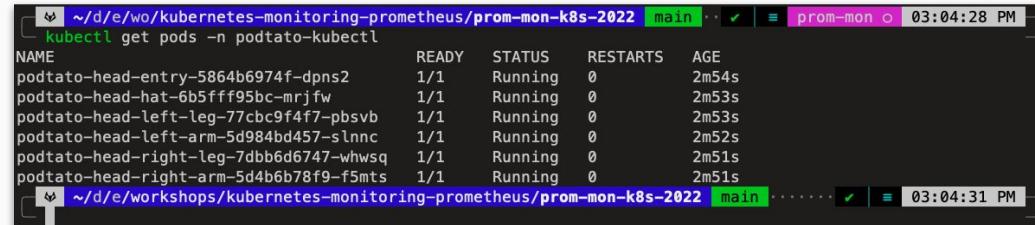
A third callout box highlights the '+ 17 alerts' button, showing the following command:

```
$ kubectl --namespace monitoring port-forward svc/alertmanager-main 30000:9093 >/dev/null &
```

EXERCISE: Trigger alerts 1/x - deploy podtato-head app

@dnsmichi 

1. Deploy app
2. Inspect pods & services
3. Add portforwarding
 - a. Open <http://localhost:40000/>



The screenshot shows two terminal windows. The top window displays the command `kubectl get pods -n podtato-kubectl` with the following output:

NAME	READY	STATUS	RESTARTS	AGE
podtato-head-entry-5864b6974f-dpns2	1/1	Running	0	2m54s
podtato-head-hat-6b5fff95bc-mrjf	1/1	Running	0	2m53s
podtato-head-left-leg-77cbc9f4f7-pbsvb	1/1	Running	0	2m53s
podtato-head-left-arm-5d984bd457-slnnc	1/1	Running	0	2m52s
podtato-head-right-leg-7dbb6d6747-whwsq	1/1	Running	0	2m51s
podtato-head-right-arm-5d4b6b78f9-f5mts	1/1	Running	0	2m51s

The bottom window shows the command `kubectl --namespace podtato-kubectl port-forward svc/podtato-head-entry 40000:9000 &`.

```
$ kubectl create -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/podtato-head/manifest.yaml

$ kubectl get pods -n podtato-kubectl
$ kubectl get services -n podtato-kubectl

# Add portforwarding to access http://localhost:40000

$ kubectl --namespace podtato-kubectl port-forward svc/podtato-head-entry 40000:9000 &
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/podtato-head/manifest.yaml>

EXERCISE: Verify DNS resolution - **tip before you proceed**

@dnsmichi 

1. Before monitoring resources, verify domains and DNS resolution
2. Follow Kubernetes docs

```
$ kubectl apply -f https://k8s.io/examples/admin/dns/dnsutils.yaml
$ kubectl get pods dnsutils

$ kubectl exec -i -t dnsutils -- nslookup kubernetes.default

$ kubectl exec -i -t dnsutils -- nslookup podtato-head-entry.podtato-kubectl.svc.cluster.local
```

```
~/d/e/wo/k/prom-mon-k8s-2022/blackbox-exporter custom-podtato-chaos t3 ?1 ..... ✓ 6s = prom-mon o 03:40:36 PM
[kubectl exec -i -t dnsutils -- nslookup podtato-head-entry.podtato-kubectl.svc.cluster.local
Server: 10.43.0.10
Address: 10.43.0.10#53

Name: podtato-head-entry.podtato-kubectl.svc.cluster.local
Address: 10.43.245.70
```

<https://kubernetes.io/docs/tasks/administer-cluster/dns-debugging-resolution/>

1. Add Blackbox Exporter Probe
2. Verify in Prometheus targets <http://localhost:10000/targets>
3. Add Grafana panels in Apps dashboard
 - a. Title: Probe Availability (%)
 - b. Query: `avg_over_time(probe_success{environment="podtato-head"}[60s:1s])*100`
 - c. Title: Probe Duration (ms)
 - d. Query: `avg_over_time(probe_duration_seconds{environment="podtato-head"}[60s:1s])*1000`

```
$ kubectl create -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o1ly-2022/-/raw/main/kubernetes/o1ly/podtato-head/probe-podtato-head.yml
```

```
$ kubectl get probes -n monitoring
```

```
$ kubectl --namespace monitoring port-forward
svc/prometheus-k8s 10000:9090 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/grafana
20000:3000 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward
svc/alertmanager-main 30000:9093 >/dev/null &
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o1ly-2022/-/raw/main/kubernetes/o1ly/podtato-head/probe-podtato-head.yml>

EXERCISE: Trigger alerts 3/x - Add grafana dashboard

@dnsmichi 

1. Open Grafana at <http://localhost:20000/>
2. Add Grafana panels in **Apps** dashboard
 - a. Datasource: Prometheus
 - b. Title: Probe Availability (%)
 - c. Query:

```
avg_over_time(probe_success{environment="podtato-head"} [60s:1s]) *100
```

- d. Title: Probe Duration (ms)
- e. Query:

```
avg_over_time(probe_duration_seconds{environment="podtato-head"} [60s:1s]) *1000
```



```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 10000:9090 >/dev/null &  
  
$ kubectl --namespace monitoring port-forward svc/grafana 20000:3000 >/dev/null &  
  
$ kubectl --namespace monitoring port-forward svc/alertmanager-main 30000:9093 >/dev/null &
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/podtato-head/probe-podtato-head.yml>

1. Add PrometheusRule for alerts

```
$ kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o1ly-2022/-/raw/main/kubernetes/alerts/podtato-head-probe-alerts.yml
```

```
apiVersion: monitoring.coreos.com/v1
kind: PrometheusRule
metadata:
  labels:
    app: kube-prometheus-stack
    name: podtato-head-probe-alerts
    namespace: monitoring
spec:
  groups:
  - name: critical-rules
    rules:
    - alert: ProbeFailing
      expr: up{job="probe/monitoring/podtato-head-blackbox-exporter", environment="podtato-head"} == 0 or
            probe_success{job="probe/monitoring/podtato-head-blackbox-exporter", environment="podtato-head"} == 0
      for: 10m
      labels:
        severity: critical
      annotations:
        summary: Podtato-head probe Down
        description: "Podtato-head probe is Down\n{{ $labels.instance }}"
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o1ly-2022/-/blob/main/kubernetes/alerts/podtato-head-probe-alerts.yml>

1. Delete podtato-head deployment
2. Trigger probe alerts
3. Alternatively, kill some pods manually

```
$ kubectl delete -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/podtato-head/manifest.yaml
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/podtato-head/manifest.yaml>

EXERCISE: Trigger alerts 6/x - Verify alerts and graphs

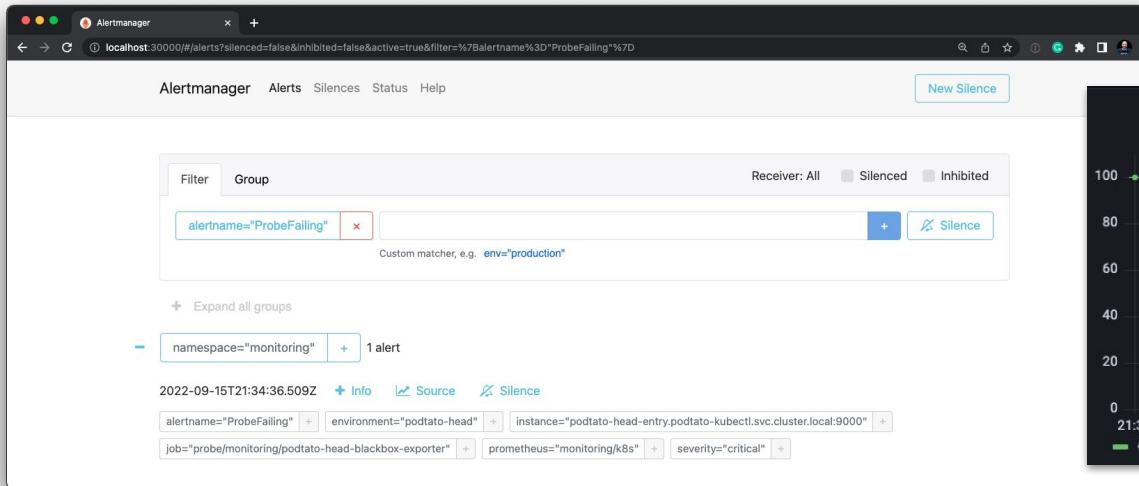
@dnsmichi 

1. Alerts: <http://localhost:30000/>
2. Graphs: <http://localhost:20000/>

```
$ kubectl --namespace monitoring port-forward svc/prometheus-k8s 10000:9090 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/grafana 20000:3000 >/dev/null &
```

```
$ kubectl --namespace monitoring port-forward svc/alertmanager-main 30000:9093 >/dev/null &
```



The screenshot shows the Alertmanager interface at <http://localhost:30000/#/alerts?silenced=false&inhibited=false&active=true&filter=%7Balertname%3D%22ProbeFailing%22>. The interface includes a filter bar with 'alertname="ProbeFailing"', a 'New Silence' button, and a 'Silence' button. Below the filter are buttons for 'namespace="monitoring"' and '+ 1 alert'. The alert details show it was triggered at 2022-09-15T21:34:36.509Z with severity 'critical'. The alert is associated with 'environment="podtato-head", instance="podtato-head-entry.podtato-kubectl.svc.cluster.local:9000", job="probe/monitoring/podtato-head-blackbox-exporter", prometheus="monitoring/k8s"'. There are also 'Info', 'Source', and 'Silence' buttons.



Service Level

Agreement - 99.5% availability

Objective - 99.9% availability

Indicator - errors, latency, ...

Error budgets

<https://landing.google.com/sre/workbook/chapters/implementing-slos/>

<https://engineering.bitnami.com/articles/implementing-slos-using-prometheus.html>

<https://grafana.com/blog/2019/11/27/kubecon-recap-how-to-include-latency-in-slo-based-alerting/>

<https://github.com/google/prometheus-slo-burn-example/blob/master/prometheus/slos.rules.yml>

<https://github.com/prometheus/prometheus/issues/6209>

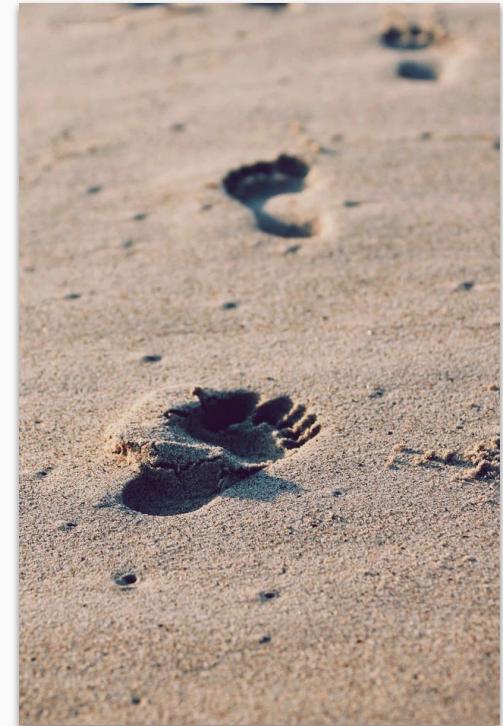


Photo by Christopher Sardegna on [Unsplash](#)

Ops Confidence

From Metrics to Alerts to SLOs

Customizations and dashboards



Kubesimplify

Workshops



@dnsmichi

Monitoring

Metrics

key/tag

Values

Alerts on PromQL queries

SLOs



Photo by Silas Baisch on [Unsplash](#)

Latency

Traffic

Errors

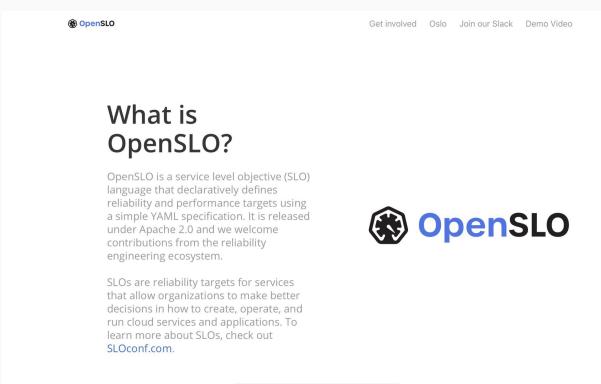
Saturation



SLO spec - OpenSLO

SLO generators - Sloth

SLO management - Pyrra



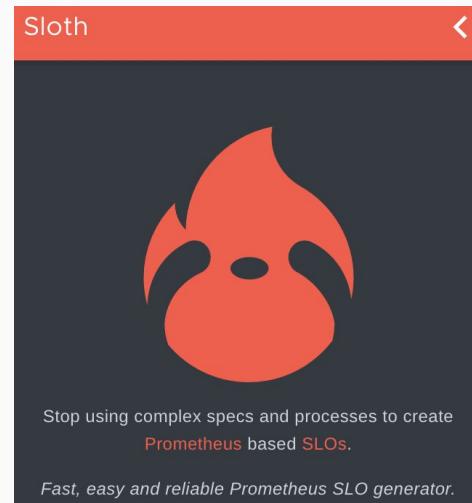
openSLO

Get involved Oslo Join our Slack Demo Video

What is OpenSLO?

OpenSLO is a service level objective (SLO) language that declaratively defines reliability and performance targets using a simple YAML specification. It is released under Apache 2.0 and we welcome contributions from the reliability engineering ecosystem.

SLOs are reliability targets for services that allow organizations to make better decisions in how to create, operate, and run cloud services and applications. To learn more about SLOs, check out [SLOconf.com](#).



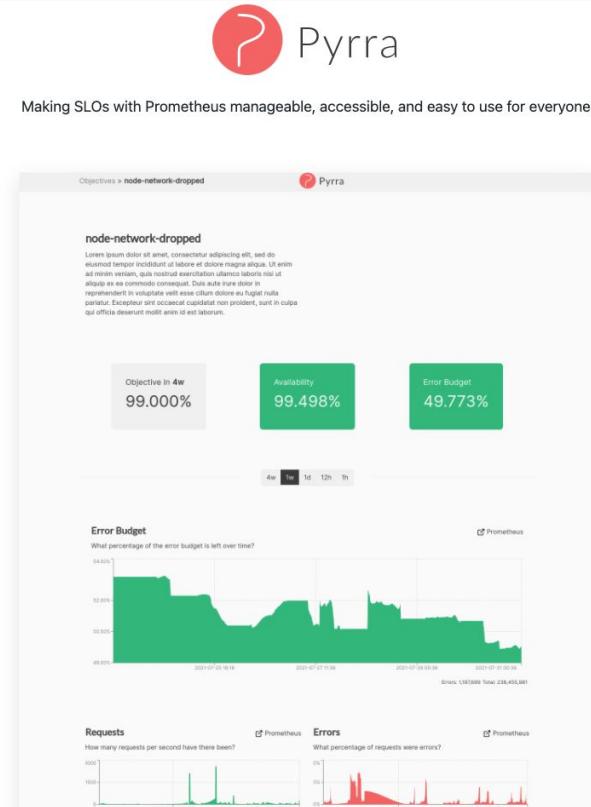
Sloth



Stop using complex specs and processes to create Prometheus based SLOs.

Fast, easy and reliable Prometheus SLO generator.

<https://openslo.com/>
<https://sloth.dev/>
<https://pyrra.dev/>



Objectives > node-network-dropped Pyrra

node-network-dropped

Loreum ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam, Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat. Duis aute irure dolor in reprehenderit in voluptate velit esse cillum dolore eu fugiat nulla pariatur. Excepteur sint occaecat cupidatat non proident, sunt in culpa qui officia deserunt mollit anim id est laborum.

Objective in 4w 99.000% Availability 99.498% Error Budget 49.773%

4w 1w 1d 1h

Error Budget

What percentage of the error budget is left over time?



Prometheus

Requests

How many requests per second have these been?



Prometheus

Errors

What percentage of requests were errors?



Prometheus

Many dashboards & alerts

Learning & Documentation

Customizations

Onboarding

Goals

Immediately see what's important during an incident



Photo by Silas Baisch on [Unsplash](#)

Use jsonnet

Develop rules and dashboards

Monitor other namespaces

Add applications

```
customize.jsonnet — my-kube-prometheus

customise.jsonnet ×
customise.jsonnet
  1 local kp = (import 'kube-prometheus/main.libsonnet') + {
  2   _config::: {
  3     namespace: 'monitoring',
  4
  5     prometheus::: {
  6       namespaces: ["default", "kube-system", "ecc"],
  7     },
  8   },
  9 }
10
11 { ['00namespace-' + name]: kp.kubePrometheus[name] for name in std.objectFields(kp.kubePrometheus) } +
12 { ['0prometheus-operator-' + name]: kp.prometheusOperator[name] for name in std.objectFields(kp.prometheusOperator) } +
13 { ['node-exporter-' + name]: kp.nodeExporter[name] for name in std.objectFields(kp.nodeExporter) } +
14 { ['kube-state-metrics-' + name]: kp.kubeStateMetrics[name] for name in std.objectFields(kp.kubeStateMetrics) } +
15 { ['alertmanager-' + name]: kp.alertmanager[name] for name in std.objectFields(kp.alertmanager) } +
16 { ['prometheus-' + name]: kp.prometheus[name] for name in std.objectFields(kp.prometheus) } +
17 { ['grafana-' + name]: kp.grafana[name] for name in std.objectFields(kp.grafana) }
18
```

<https://prometheus-operator.dev/docs/developing-prometheus-rules-and-grafana-dashboards/>

<https://prometheus-operator.dev/docs/kube/monitoring-other-namespaces/>

<https://github.com/prometheus-operator/kube-prometheus#customizing-kube-prometheus>

<https://docs.bitnami.com/tutorials/implementing-slos-using-prometheus>

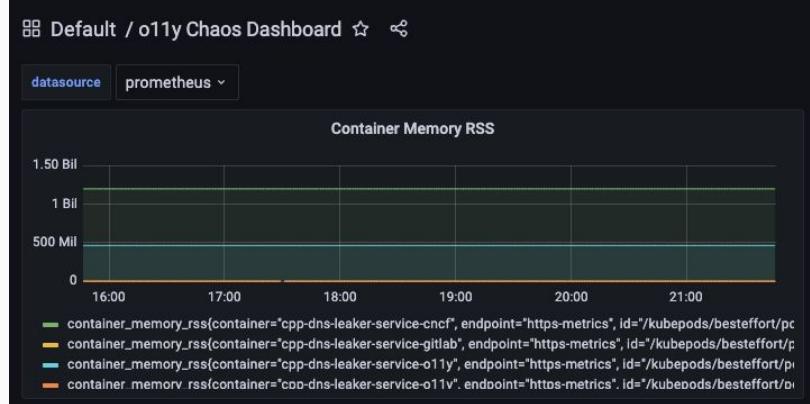
Custom dashboards

@dnsmichi 

 o11y-chaos-dashboard.jsonnet  857 bytes

```
1 local grafana = import 'grafonnet/grafana.libsonnet';
2 local dashboard = grafana.dashboard;
3 local row = grafana.row;
4 local prometheus = grafana.prometheus;
5 local template = grafana.template;
6 local graphPanel = grafana.graphPanel;
7
8 {
9   'o11y-chaos-dashboard.json':
10   dashboard.new('o11y Chaos Dashboard')
11   .addTemplate(
12     {
13       current: {
14         text: 'Prometheus',
15         value: 'Prometheus',
16       },
17       hide: 0,
18       label: null,
19       name: 'datasource',
20       options: [],
21       query: 'prometheus',
22       refresh: 1,
23       regex: '',
24       type: 'datasource',
25     },
26   )
27   .addRow(
28     row.new()
29     .addPanel(graphPanel.new('Container Memory RSS', span=6, datasource='$datasource')
30               .addTarget(prometheus.target('container_memory_rss{container=~"cpp-dns-leaker-service.*"}')))
31   ),
32 }
```

```
31   },
32   dashboards++: (import 'o11y-chaos-dashboard.jsonnet'),
33 },
```



<https://prometheus-operator.dev/docs/developing-prometheus-rules-and-grafana-dashboards/>

<https://github.com/prometheus-operator/kube-prometheus#customizing-kube-prometheus>

<https://gitlab.com/everyonecancontribute/observability/k8s-o11y-chaos/-/blob/main/kubernetes/o11y/my-kube-prometheus/o11y-chaos-dashboard.jsonnet>

Beyond Metrics



Kubesimplify

Workshops



Photo by [Michael Friedrich](#)

Metrics

Logs/Events

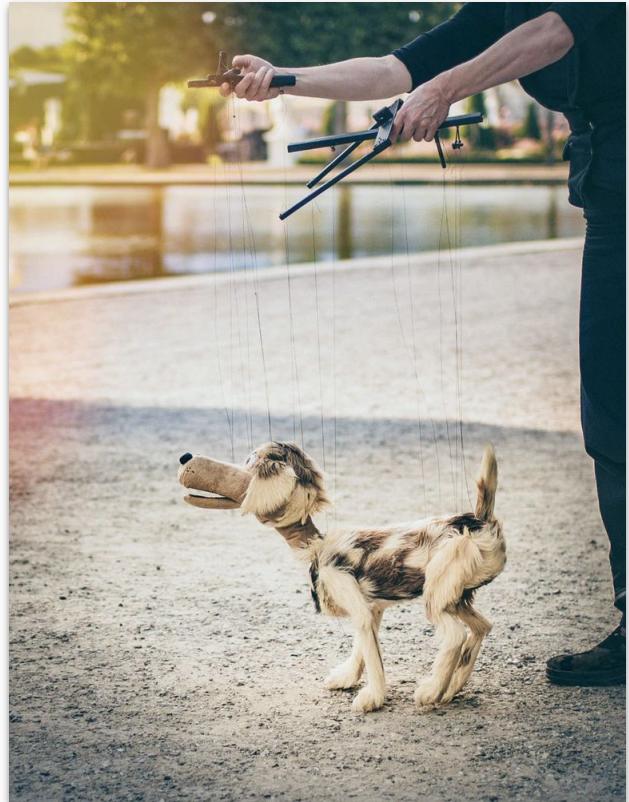
Traces

Profiling

Error tracking

<more event types>

... is it Observability?



Central log management evaluation

Self-managed

Elastic/OpenSearch, Beats agents

Grafana Loki/promtail

Fluentbit/vector as sidecars

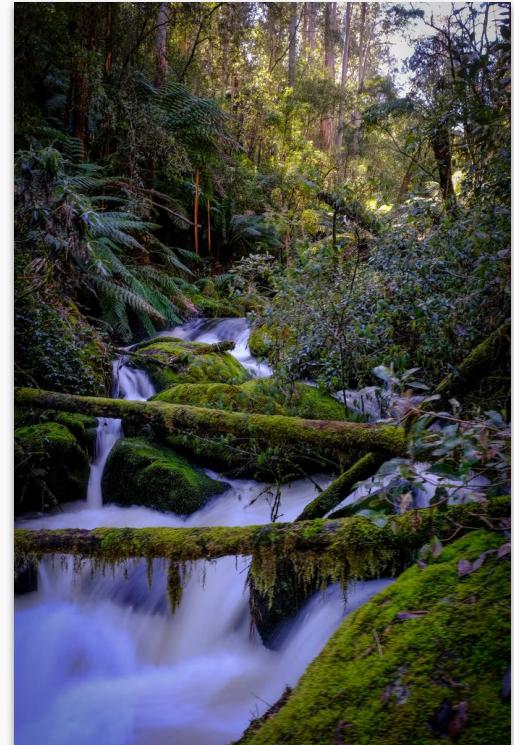
... or SaaS

Datadog, Splunk, Dynatrace, etc.

What logs do we really need?

<https://blog.flant.com/logs-in-kubernetes-expectations-vs-reality/>

Photo by [Zac Porter](#) on [Unsplash](#)



Traces, like logs

Spans with start/end time

Context

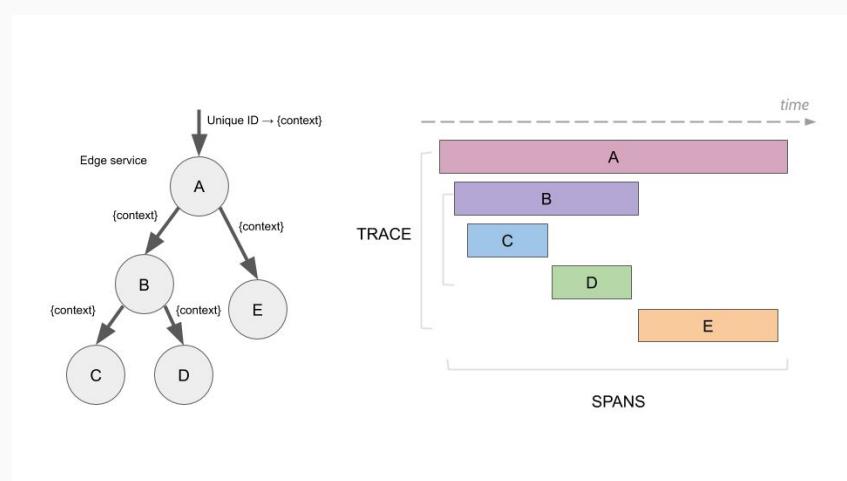
Metadata enrichment

Instrumentation

Code changes by developers

Automated

Distributed environments



<https://www.jaegertracing.io/docs/1.25/architecture/>

<https://github.com/grafana/tempo/tree/main/example/helm>

<https://github.com/open-telemetry/opentelemetry-operator>

<https://docs.google.com/presentation/d/1MAVFeSsTNVWC9wPGOlglg83wh8GFtR9hPbdVHuutgOWA/edit>

Image credit:
[Jaeger Tracing Docs](#)

Specification and framework beyond vendors

Collector + Sidecar

Bring your own backend

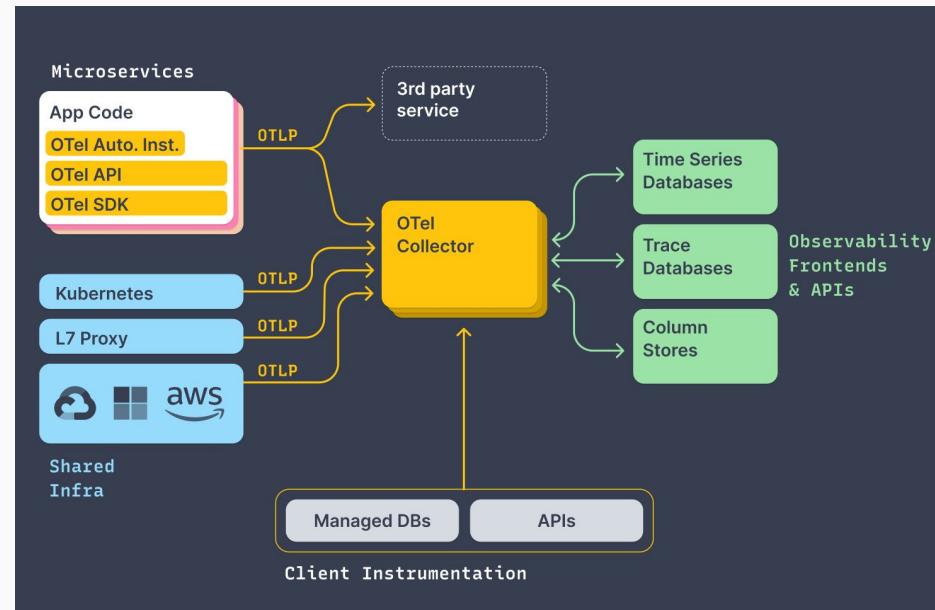
Jaeger for Traces

Prometheus for Metrics

Client libraries & SDKs

Manual instrumentation (C++, Go, etc.)

Auto-instrumentation (Java)



Kubernetes

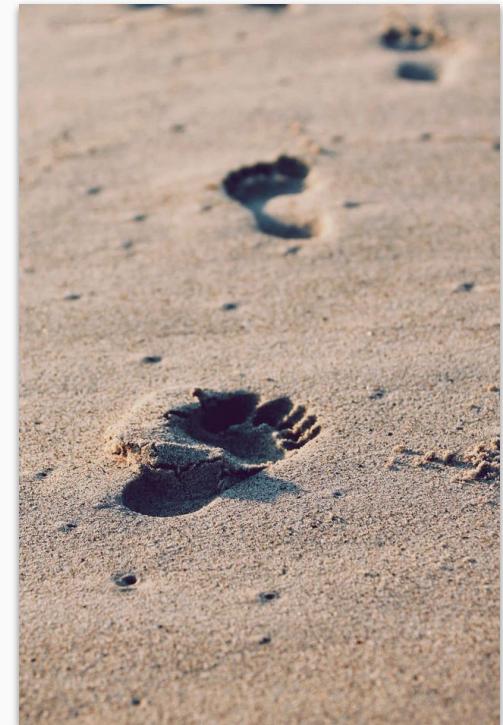
Components send traces

App instrumentation with traces

Send to OpenTelemetry collector

Store in Jaeger, etc.

Visualize, correlate, alert



Traces help analyse slow requests

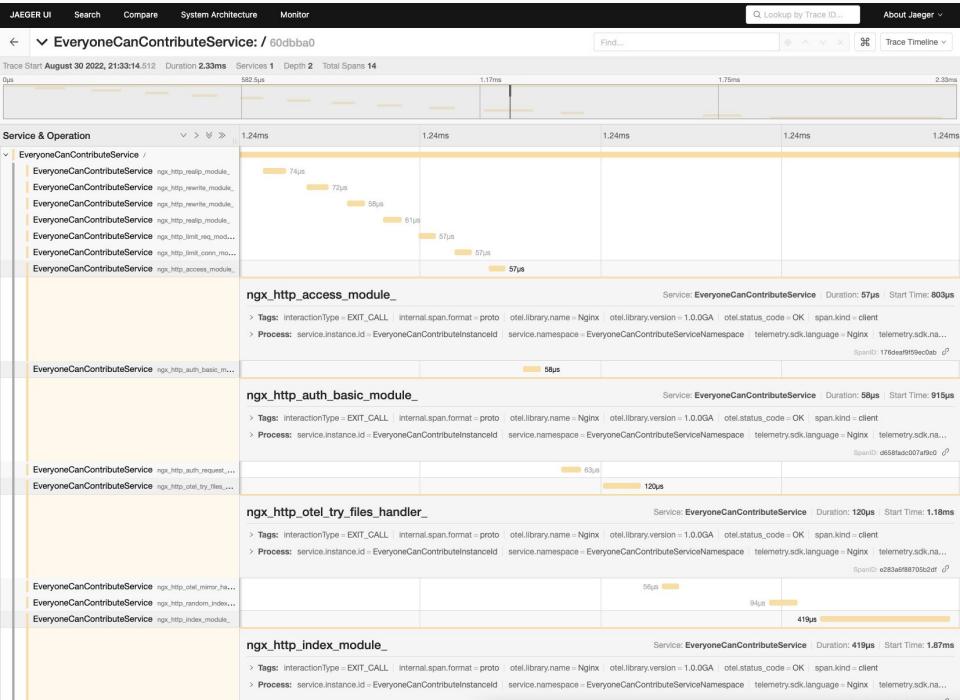
@dnsmichi 

Client requests data from HTTP server

Server asks backend, collects data, creates client response

Instrument Nginx/Apache to see traces

OpenTelemetry webserver SDK



<https://gitlab.com/everyonecancontribute/observability/nginx-opentelemetry>

EXERCISE: Deploy OpenTelemetry demo

@dnsmichi 

1. Helm chart installs the OpenTelemetry Collector and Jaeger Tracing
 - a. Includes a shop and load generator demo
2. Add port-forwarding

```
helm repo add open-telemetry
https://open-telemetry.github.io/opentelemetry-helm-charts
```

```
helm install my-otel-demo open-telemetry/opentelemetry-demo
```

Frontend UI <http://localhost:8080>

```
$ kubectl port-forward svc/my-otel-demo-frontend
8080:8080 >/dev/null &
```

Jaeger UI <http://localhost:16686>

```
$ kubectl port-forward svc/my-otel-demo-jaeger
16686:16686 >/dev/null &
```

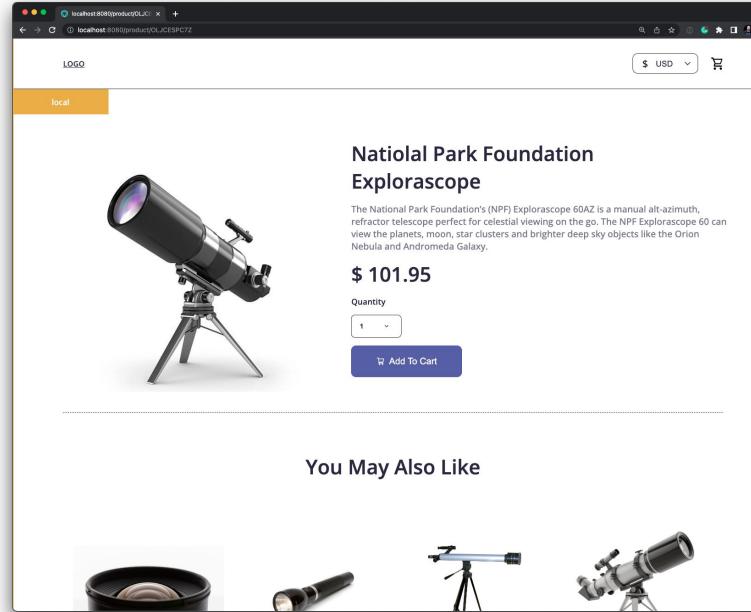
Locust UI <http://localhost:8089>

```
$ kubectl port-forward svc/my-otel-demo-loadgenerator
8089:8089 >/dev/null &
```

EXERCISE: Inspect the shop website

@dnsmichi 

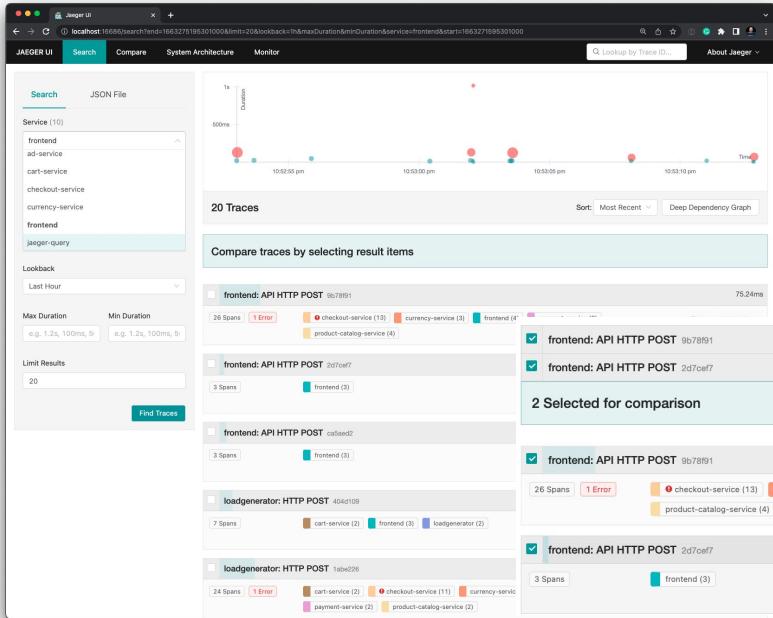
1. Navigate into the shop website in <http://localhost:8080>
 - a. Try different paths and checkouts



EXERCISE: Analyse traces in Jaeger

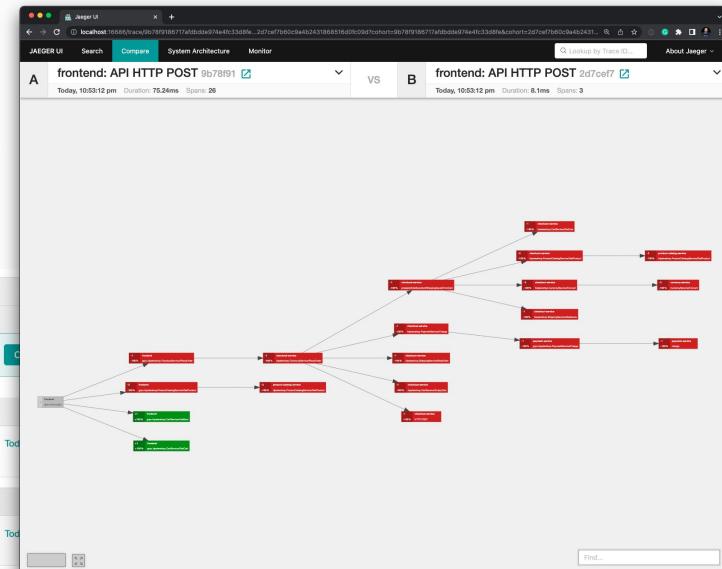
@dnsmichi 

1. Open the Jaeger UI <http://localhost:16686> and select the `frontend` service
2. Analyse different services and compare traces



The screenshot shows the Jaeger UI search results page. On the left, there's a sidebar with a search bar and dropdowns for 'Service' (set to 'frontend') and 'Lookback' (set to 'Last Hour'). Below these are filters for 'Max Duration' and 'Min Duration'. A 'Limit Results' dropdown is set to '20'. At the bottom right of the sidebar is a 'Find Traces' button. The main area displays a timeline chart showing several trace segments over a 10-second period. Below the chart is a section titled '20 Traces' with a 'Sort: Most Recent' dropdown and a 'Deep Dependency Graph' button. A list of traces is shown, each with a checkbox and a brief description. The first trace is selected:

- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)
- frontend: API HTTP POST 9b78f91 (75.24ms)
- frontend: API HTTP POST 2d7cef7 (75.24ms)



Discussion: Why traces?

@dnsmichi 

Distributed microservices in your Kubernetes cluster

Metrics won't tell you that the Redis caching for one load balancer route is slow

Pod logs with errors at a time, but what is the root cause?

Customers are not able to buy products, or navigate the website

Traces provide context and metadata, e.g. software versions, dependencies, etc.

Developers add code instrumentation, and can make their lives easier with later debugging a trace ... rather than blind guesses from metrics and logs.

Review your code and deployed applications

Look into OpenTelemetry instrumentation libraries

Evaluate auto-instrumentation

Focus on traces

OpenTelemetry collector can receive more types

Iterate on metrics, logs, etc. later

Evaluate tracing integration into DevOps and SRE workflows

1. There isn't an "easy" exercise for logging :-(
2. Sidecars, DaemonSets, node, etc. strategies
3. Run log management inside Kubernetes
 - a. Elastic Cloud
 - b. Grafana Loki/promtail
4. Use sidecars/agents (fluentd) to collect logs and send to platforms
 - a. SaaS: Logz.io, Datadog, Dynatrace, Splunk, Honeycomb, Elastic, etc.
 - b. Self-managed: Elastic/Opensearch, Graylog, Grafana Loki, etc.
5. Vendor websites are overwhelming with Observability "everything"
6. No recommendations here - this needs evaluation, step by step

Application Performance Insights

Continuous Profiling

Alerting?

SLOs?



<https://www.parca.dev/>

<https://twitter.com/fredbrancz/status/1447655522979692555>

<https://www.polarsignals.com/>

<https://everyonecancontribute.com/post/2021-06-02-cafe-32-polar-signals-continuous-profiling/>

Performance Scaling

Strategies and ideas



Kubesimplify

Workshops



Photo by [Michael Friedrich](#)



@dnsmichi

Metrics

Retention, Compaction

Traces

Sampling

Logs

Archiving

“What timeframe do you need for debugging incidents?”

Compliance requirements: “Archive everything on a magnet tape”

Thanos for High Availability and long term storage

Cortex for distributed clusters, aggregated queries over Prometheus nodes



Thanos

<https://thanos.io/tip/thanos/design.md/>
<https://cortexmetrics.io/docs/architecture/>
<https://prometheus-operator.dev/docs/operator/thanos/>
<https://github.com/thanos-io/kube-thanos/>



cortex

Monitoring and Observability tooling in your cluster requires additional resources.

<https://www.opencost.io/> and others can help estimate costs.

Persistent volumes plan ahead: How much storage will metrics/traces/logs need?

Manage everything yourself, or send data to SaaS platforms?

Multi cluster approach

Prometheus stack in each cluster?

Enterprise requirements

RBAC

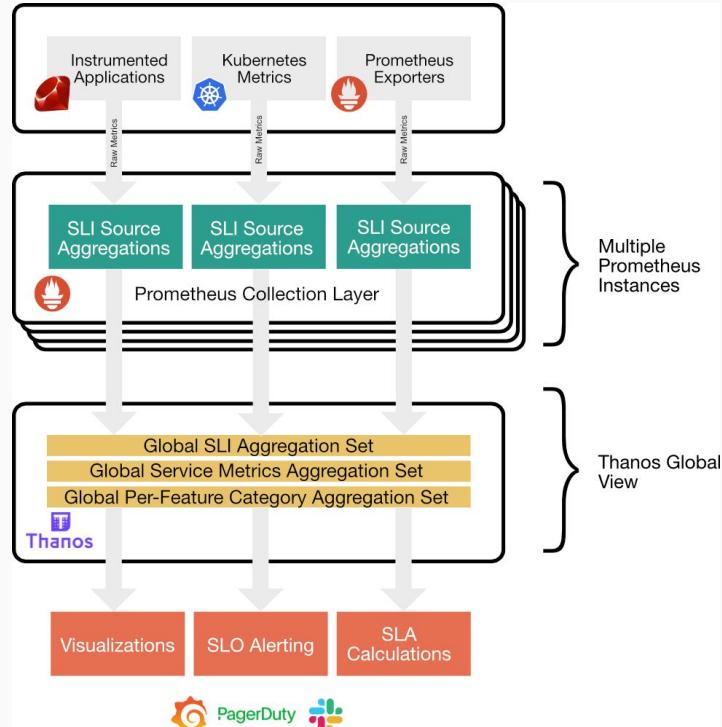
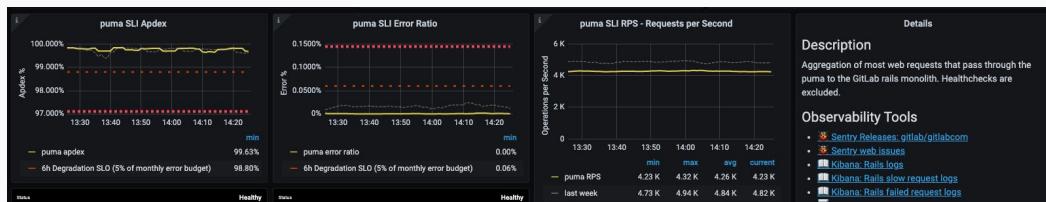
SSO

“Central” observability with SaaS platforms

Prometheus remote_write

OpenTelemetry collectors that send to SaaS (Datadog agent for example)

- Define Metrics Catalog
- Generate alerts and dashboards
 - Key metrics
 - Thresholds
 - Description + Observability tool URLs



💡 How We are Dealing with Metrics at Scale on GitLab.com - Andrew Newdigate, GitLab

<https://www.youtube.com/watch?v=6sfr2IGJQXk>

<https://about.gitlab.com/handbook/engineering/monitoring/>

<https://dashboards.gitlab.net/d/web-main/web-overview?from=now-1h&orgId=1&to=now>

<https://gitlab.com/gitlab-com/runbooks/-/tree/master/metrics-catalog>

<https://gitlab.com/gitlab-com/ql-infra/gitlab-monitoring/-/tree/main/gitlab-monitoring>



Observability and Chaos Engineering

Automate breaking things

Benefit from confidence with alerting
and Observability



Kubesimplify

Workshops



Photo by [Michael Friedrich](#)

Simulate incidents?

@dnsmichi 

Simulating production incidents is hard 🤔

Add automated chaos 💡

Staging environments

Production deployments

Trigger alerts and SLOs 🔥

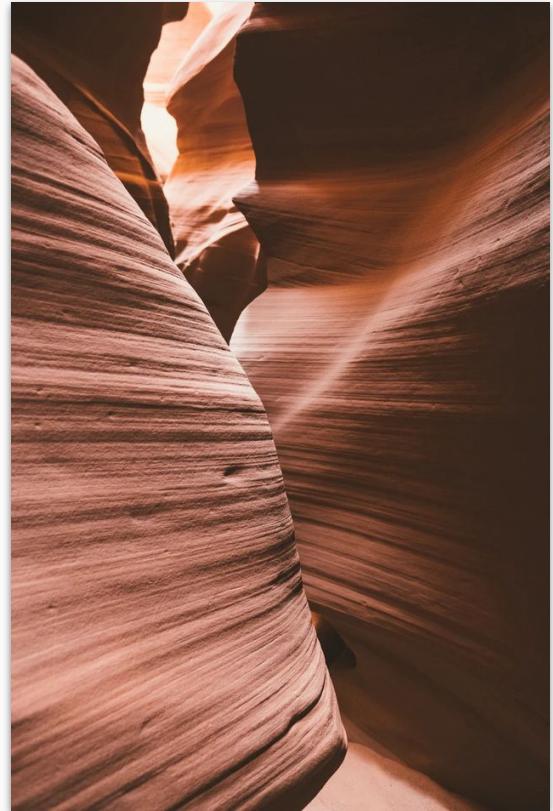


Photo by [Matteo Di Iorio](#) on [Unsplash](#)

As a SRE/Dev/Ops/DevOps, where to?

Cloud Native

Clusters

Deployments

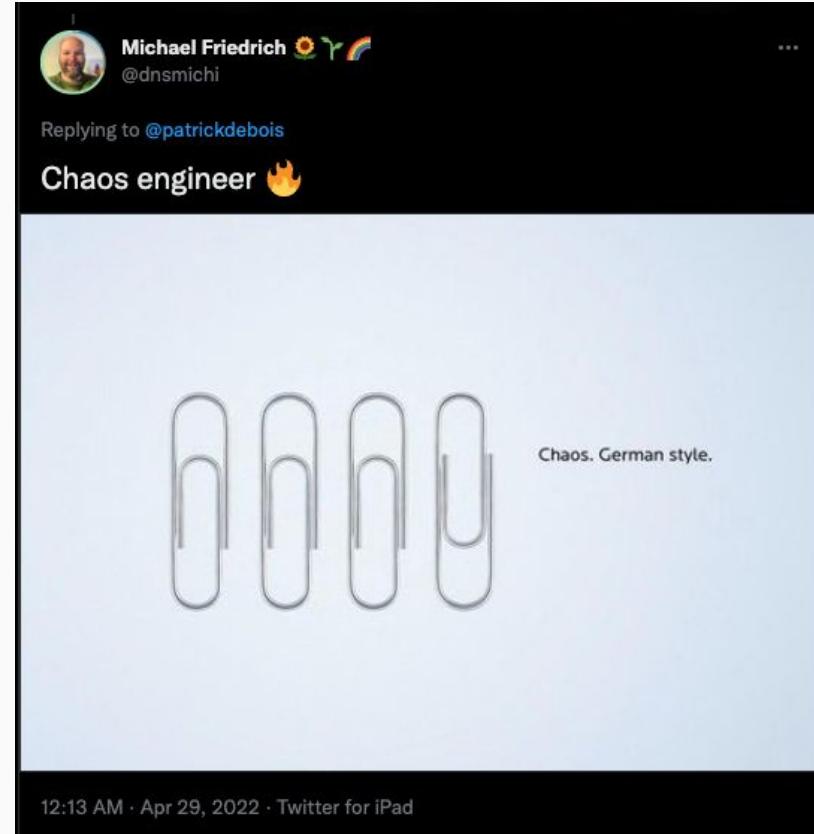
Chaos Framework

Experiments

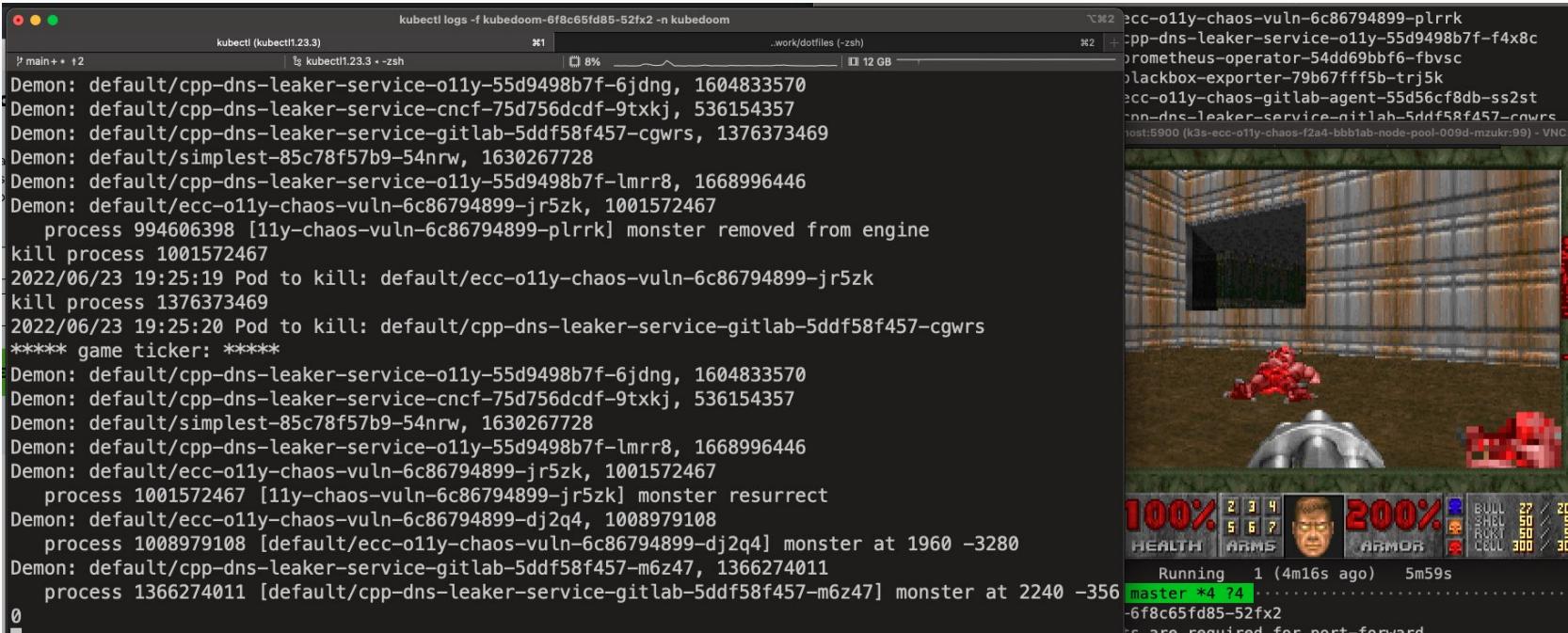
Instrumentation SDKs

<https://o11y.love/topics/chaos-engineering/>

@dnsmichi 



Kubedoom (might not be applicable for \$work)



The terminal window displays logs from several Kubernetes pods:

- Demon: default/cpp-dns-leaker-service-o11y-55d9498b7f-6jdng, 1604833570
- Demon: default/cpp-dns-leaker-service-cncf-75d756dcdf-9txkj, 536154357
- Demon: default/cpp-dns-leaker-service-gitlab-5ddf58f457-cgwrs, 1376373469
- Demon: default/simplest-85c78f57b9-54nrw, 1630267728
- Demon: default/cpp-dns-leaker-service-o11y-55d9498b7f-lmrr8, 1668996446
- Demon: default/ecc-o11y-chaos-vuln-6c86794899-jr5zk, 1001572467
process 994606398 [11y-chaos-vuln-6c86794899-plrrk] monster removed from engine
- kill process 1001572467
- 2022/06/23 19:25:19 Pod to kill: default/ecc-o11y-chaos-vuln-6c86794899-jr5zk
- kill process 1376373469
- 2022/06/23 19:25:20 Pod to kill: default/cpp-dns-leaker-service-gitlab-5ddf58f457-cgwrs
- ***** game ticker: *****
- Demon: default/cpp-dns-leaker-service-o11y-55d9498b7f-6jdng, 1604833570
- Demon: default/cpp-dns-leaker-service-cncf-75d756dcdf-9txkj, 536154357
- Demon: default/simplest-85c78f57b9-54nrw, 1630267728
- Demon: default/cpp-dns-leaker-service-o11y-55d9498b7f-lmrr8, 1668996446
- Demon: default/ecc-o11y-chaos-vuln-6c86794899-jr5zk, 1001572467
process 1001572467 [11y-chaos-vuln-6c86794899-jr5zk] monster resurrect
- Demon: default/ecc-o11y-chaos-vuln-6c86794899-dj2q4, 1008979108
process 1008979108 [default/ecc-o11y-chaos-vuln-6c86794899-dj2q4] monster at 1960 -3280
- Demon: default/cpp-dns-leaker-service-gitlab-5ddf58f457-m6z47, 1366274011
process 1366274011 [default/cpp-dns-leaker-service-gitlab-5ddf58f457-m6z47] monster at 2240 -356

The terminal also shows a status bar with tabs for .work/dotfiles (~zsh) and .work/dotfiles (~zsh). The background of the terminal window features a screenshot of the video game Doom.

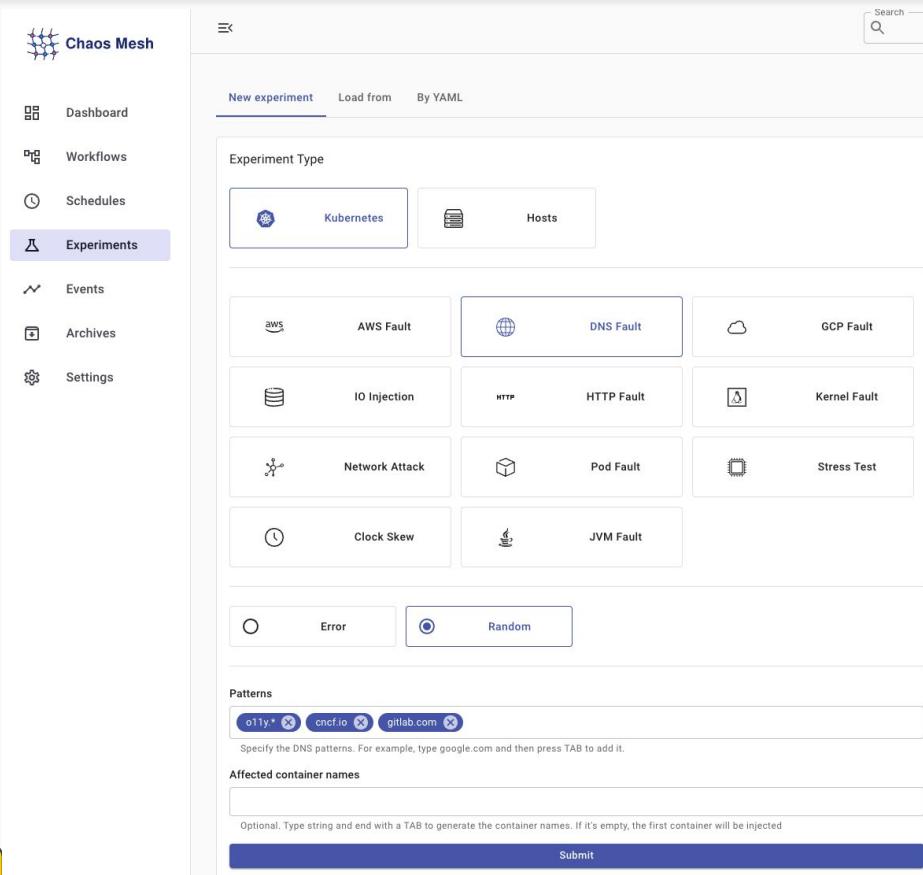
Fail Kubernetes or Hosts

Pods, network, http, time, DNS, ...

Run Chaos Experiments

Once

Continuously as schedule



The screenshot shows the Chaos Mesh web interface. On the left is a sidebar with icons for Dashboard, Workflows, Schedules, Experiments (which is selected and highlighted in blue), Events, Archives, and Settings. The main area has tabs for 'New experiment', 'Load from', and 'By YAML'. Below is a section for 'Experiment Type' with two tabs: 'Kubernetes' (selected) and 'Hosts'. A grid of fault types is shown:

 AWS Fault	 DNS Fault	 GCP Fault
 IO Injection	 HTTP Fault	 Kernel Fault
 Network Attack	 Pod Fault	 Stress Test
 Clock Skew	 JVM Fault	

Below the grid are buttons for 'Error' and 'Random'. A 'Patterns' section contains 'o11y+', 'cncf.io', and 'gitlab.com'. A note says 'Specify the DNS patterns. For example, type google.com and then press TAB to add it.' A 'Affected container names' input field is present, with a note: 'Optional. Type string and end with a TAB to generate the container names. If it's empty, the first container will be injected'. A 'Submit' button is at the bottom.

<https://chaos-mesh.org/docs/quick-start/>

<https://chaos-mesh.org/docs/simulate-dns-chaos-on-kubernetes/>

Generate some chaos

@dnsmichi 

{...} pod-fail-schedule.yaml 440 bytes

```
1 kind: Schedule
2 apiVersion: chaos-mesh.org/v1alpha1
3 metadata:
4   namespace: default
5   name: pod-kill-cpp-dns-leaker
6 spec:
7   schedule: '* * * * *'
8   startingDeadlineSeconds: null
9   concurrencyPolicy: Forbid
10  historyLimit: 1
11  type: PodChaos
12  podChaos:
13    selector:
14      namespaces:
15        - default
16      labelSelectors:
17        app: cpp-dns-leaker-service-o11y
18    mode: all
19    action: pod-failure
20    duration: 10s
21    gracePeriod: 0
```



NAME	READY	STATUS	RESTARTS	AGE
simplest-85c78f57b9-4z7z4	1/1	Running	1 (6d10h ago)	8d
cpp-dns-leaker-service-gitlab-546b4bdff9-lvsn8	1/1	Running	0	41d
ecc-o11y-chaos-vuln-8646c57fc5-brzww	1/1	Running	0	41d
cpp-dns-leaker-service-cncf-d9b9898f7-zq6gp	1/1	Running	0	41d
testcurl	1/1	Running	0	25h
nginx-otel-service-c9c56c87c-bz78r	1/1	Running	0	24h
promtail-ncckw	1/1	Running	1 (31h ago)	49d
dnsutils	1/1	Running	25 (11m ago)	25h
promtail-p4f55	0/1	Running	1 (6d10h ago)	49d
promtail-p8x95	0/1	Running	0	49d
cpp-dns-leaker-service-o11y-f44459b87-klz8m	0/1	CrashLoopBackOff	7 (2s ago)	39d
cpp-dns-leaker-service-o11y-f44459b87-2vbl2	0/1	CrashLoopBackOff	8 (2s ago)	8d

Is this the “correct” chaos?

(we can see that Kubernetes heals itself)

<https://gitlab.com/everyonecancontribute/observability/k8s-o11y-chaos/-/blob/main/kubernetes/chaos/pod-fail-schedule.yaml>

<https://songraq.github.io/operation/how-to-alert-for-Pod-Restart-OOMKilled-in-Kubernetes/>

Use case: DNS errors cause memory leaks

App creates receive buffer

DNS resolve fails, or empty result

Memory leak, missing buffer delete

```
38 while (true) {  
39     // pre-allocated receive buffer  
40     char *rcvbuf = new char[1024*1024];  
41  
42     boost::asio::ip::tcp::resolver::results_type results = resolver.resolve(hostname, "443", error);  
43  
44     // DNS resolution failed or has empty results, handle the error  
45     if (error != boost::system::errc::success || results.empty()) {  
46         handle_dns_error(error);  
47         continue;  
48     }
```



1. Install Chaos Mesh on Civo (watch the trainer for this specific install)

```
helm repo add chaos-mesh https://charts.chaos-mesh.org  
  
helm search repo chaos-mesh  
  
kubectl create ns chaos-testing  
  
helm install chaos-mesh chaos-mesh/chaos-mesh \  
--namespace=chaos-testing \  
--version 2.3.1 \  
--set dnsServer.create=true \  
--set dashboard.create=true \  
--set dashboard.securityMode=false \  
--set chaosDaemon.hostNetwork=true \  
--set chaosDaemon.runtime=containerd \  
--set chaosDaemon.socketPath=/run/k3s/containerd/containerd.sock  
  
kubectl get pods --namespace chaos-testing -l app.kubernetes.io/instance=chaos-mesh  
  
kubectl --namespace chaos-testing port-forward svc/chaos-dashboard 2333
```

<https://chaos-mesh.org/docs/production-installation-using-helm/>

Follow <https://github.com/civo/kubernetes-marketplace/blob/master/chaos-mesh/install.sh>
to apply Civo specific installation parameters for k3s.

1. Deploy apps
2. Add alerting rules
3. Start port-forward and open Chaos Mesh UI at <http://localhost:2333>

```
# App deployments that try to resolve o11y.love,cncf.io,gitlab.com,kubesimplify.com
kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/cpp-dns-leaker/deployment.yaml

# Prometheus alerting rule for Container Memory RSS > 10MB
kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/cpp-dns-leaker/prometheus-rule-memory.yml

# Chaos Mesh UI port-forward
kubectl --namespace chaos-testing port-forward svc/chaos-dashboard 2333
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/cpp-dns-leaker/deployment.yaml>

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/cpp-dns-leaker/prometheus-rule-memory.yml>

EXERCISE (advanced): DNS Chaos with Chaos Mesh

@dnsmichi 

1. Create a new schedule, By YAML
 2. Copy the YAML schedule
 3. Verify affected pods and submit
 4. Attach to deployment logs

```
kubectl logs -f  
deployment/cpp-dns-leaker-service-k  
ubesimplify
```

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/chaos/dns-schedule.yaml>
<https://opensource.com/article/21/6/chaos-mesh-kubernetes>

New Schedule Load from By YAML

Create an experiment or schedule by filling in or uploading YAML

```
apiVersion: chaos-mesh.org/v1alpha1
kind: Schedule
metadata:
  name: schedule-dns-chaos-demo
spec:
  schedule: '* * * * *'
  historyLimit: 2
  concurrencyPolicy: 'Allow'
  type: 'DNSChaos'
  dnsChaos:
    action: error # important: error, no random responses
    mode: all
    patterns:
      - oilly.*
      - cncf.io
      - gitlab.com
      - containerdays.io
      - kubesimplify.com
  selector:
    namespaces:
      - default
duration: '60s'
```

es/chaos/dns-s

cpp-dns-leaker-service-kubesimplify-7b79fd48d6-n6pl2	default
--	---------

CIO

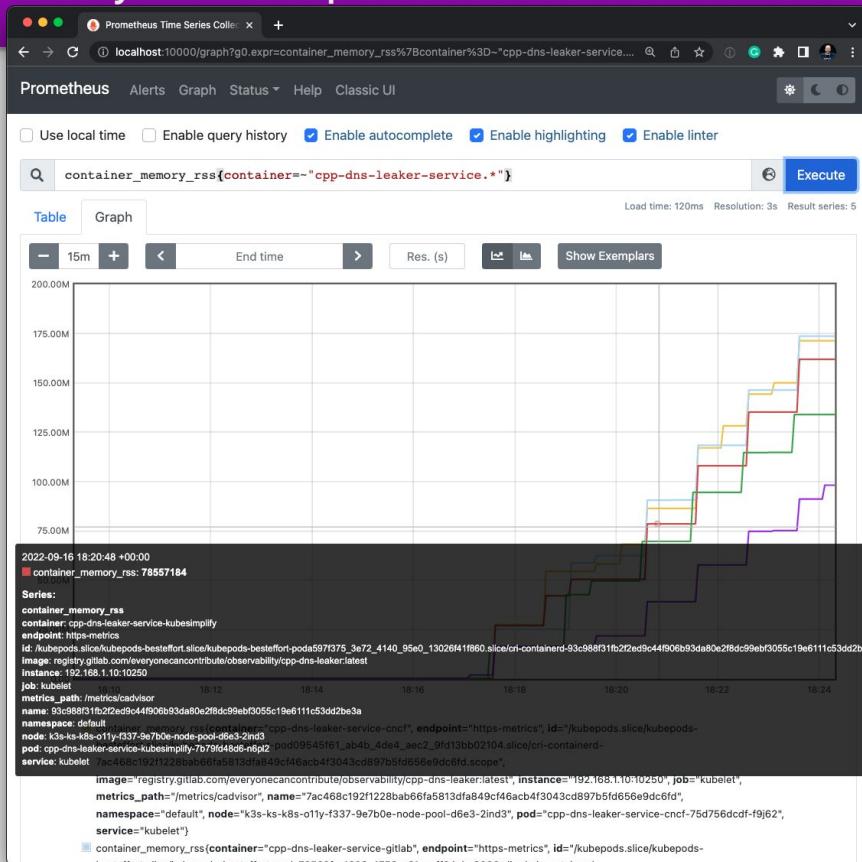


EXERCISE (advanced): Verify DNS errors and Memory Consumption

@dnsmichi 

1. Attach to deployment logs
2. Open <http://localhost:10000>
 - a. Query

```
kubectl logs -f  
deployment/cpp-dns-leaker-service-kubesimplify  
  
# Port forwarding  
kubectl --namespace monitoring port-forward  
svc/prometheus-k8s 10000:9090 >/dev/null &  
  
# PromQL query  
  
container_memory_rss{container=~"cpp-dns-leaker-  
service.*"}
```



<https://gitlab.com/everyonecancontribute/observability/cpp-dns-leaker>

More details in my Kubecon EU talk: <https://www.youtube.com/watch?v=BkREMg8adal>

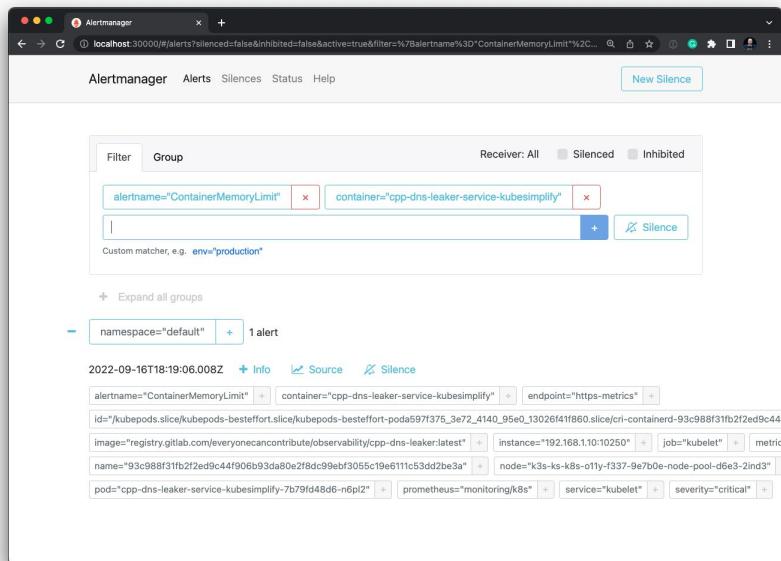
EXERCISE (advanced): Verify Alerts

@dnsmichi 

1. Open the Alert Manager at <http://localhost:30000>
2. Filter by alertname="ContainerMemoryLimit"

```
# Port forwarding

kubectl --namespace monitoring
port-forward svc/alertmanager-main
30000:9093 >/dev/null
```



The screenshot shows the Alertmanager interface running locally. The URL in the address bar is `localhost:30000#alerts?silenced=false&inhibited=false&active=true&filter=%7Balertname%3DContainerMemoryLimit%2C...`. The interface has tabs for Alertmanager, Alerts, Silences, Status, and Help. A "New Silence" button is visible in the top right. The main area shows a filter panel with two selected filters: "alertname='ContainerMemoryLimit'" and "container='cpp-dns-leaker-service-kubesimplify'". Below the filters is a "Custom matcher, e.g. env='production'" input field. A "Silence" button is also present in the filter panel. Below the filter panel, a list of alerts is shown under a "namespace='default'" section. One alert is expanded, showing detailed metrics and labels. The expanded alert details include:

- Timestamp: 2022-09-16T18:19:06.008Z
- Labels:
 - alertname="ContainerMemoryLimit"
 - container="cpp-dns-leaker-service-kubesimplify"
 - endpoint="https-metrics"
 - id="/kubepods.slice/kubepods-besteffort.slice/kubepods-besteffort-poda597f376_3e72_4140_95e0_13026141f860.slice/cr-containerd-93c988f31fb212ed9c44
 - image="registry.gitlab.com/everyonecancontribute/observability/cpp-dns-leaker:latest"
 - instance="192.168.1.10:10250"
 - job="kubelet"
 - metric_name="93c988f31fb212ed9c44f906b93da80e2f8d99ebf3055c196611c53dd2be3a"
 - node="k3s-ks-k8s-011y-f337-9e700e-ncke-pool-06e3-2ind3"
 - pod="cpp-dns-leaker-service-kubesimplify-7b7fd48d6+6pl2"
 - prometheus="monitoring/k8s"
 - service="kubelet"
- Severity: critical

<https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022-/blob/main/kubernetes/o11y/cpp-dn-s-leaker/prometheus-rule-memory.yml>

Whats next?

Alert counts

Grouping?

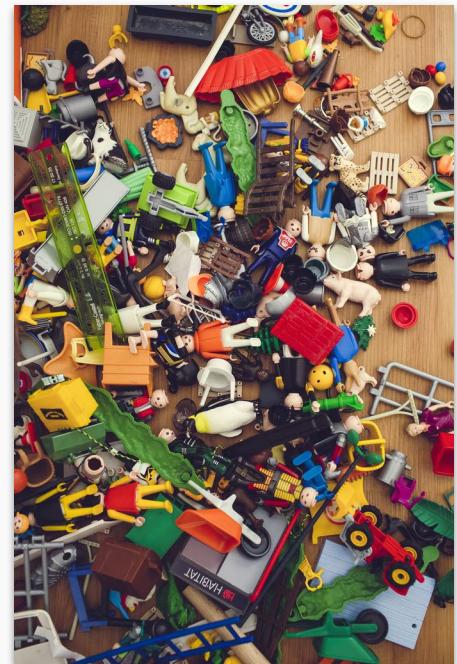
Additional context in alerts

Focus in dashboards

More context with URLs, docs

Correlate data

Reduce the amount of visible data



Traces show slow requests: Chaos

@dnsmichi 

How to get slow HTTP requests?

Add `sleep(rand(10))` to your code and deploy ... to prod with a feature flag?

Maybe. If you are crazy like me (code reviews will reject the idea ;-))

Better: Chaos experiments for HTTP, network, CPU/Memory stress tests

<https://chaos-mesh.org/docs/simulate-http-chaos-on-kubernetes/>
<https://chaos-mesh.org/docs/simulate-heavy-stress-on-kubernetes/>

```
stress-cpu-mem-nginx-otel.yaml 514 bytes

1 kind: Schedule
2 apiVersion: chaos-mesh.org/v1alpha1
3 metadata:
4   namespace: default
5   name: stress-cpu-mem-nginx-otel
6 spec:
7   schedule: '* * * * *'
8   startingDeadlineSeconds: null
9   concurrencyPolicy: Forbid
10  historyLimit: 1
11  type: StressChaos
12  stressChaos:
13    selector:
14      namespaces:
15        - default
16      labelSelectors:
17        app: nginx-otel-service
18    mode: all
19    stressors:
20      memory:
21        workers: 10
22        size: '100'
23      cpu:
24        workers: 10
25        load: 100
26        duration: 60s
```

Traces show slow requests: Chaos

@dnsmichi 

Chaos Mesh

Stress test CPU & memory

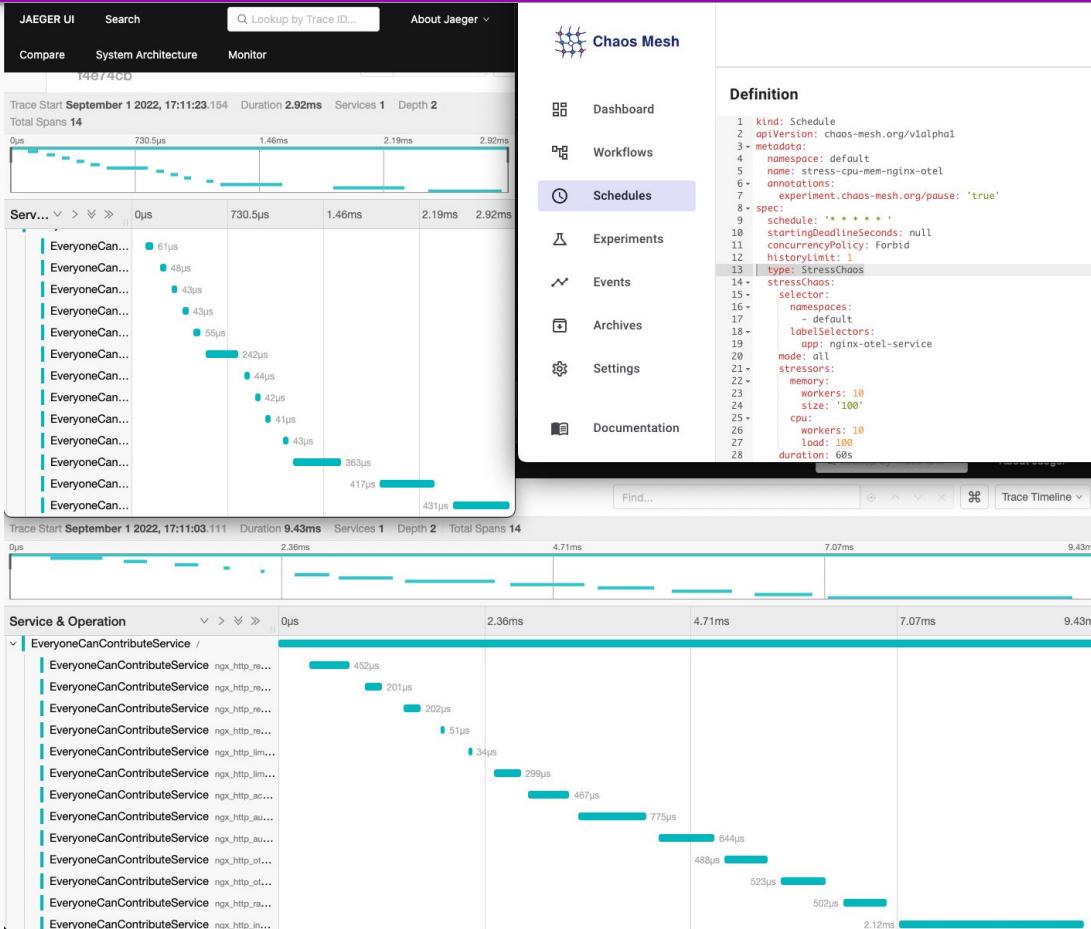
LabelSelector

app: nginx-otel-service

Jaeger

Trace time++

<https://gitlab.com/everyonecancontribute/observability/k8s-o11y-chaos/-/blob/main/kubernetes/chaos/stress-cpu-mem-nginx-otel.yml>



Exemplars

Link metrics with traces

Aggregated trace metrics

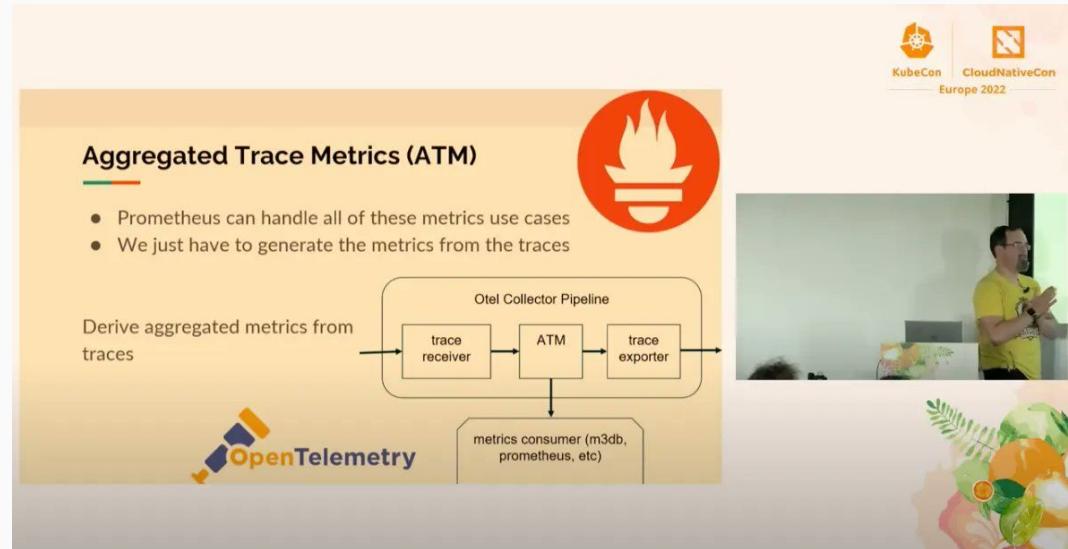
Create metrics from traces

Alerts and Chaos

Traces for Kubernetes System Components

Prometheus and Jaeger: A Match Made in Heaven! <https://www.youtube.com/watch?v=EvzTFUMqCXY>

Jaeger; Present and future, with Aggregated trace metrics: https://www.youtube.com/watch?v=NWQD_uCGP6k
<https://kubernetes.io/docs/concepts/cluster-administration/system-traces/>



Future Observability



Kubesimplify

Workshops



@dnsmichi

Photo by [Michael Friedrich](#)

"Instead of relying on static counters and gauges exposed by the operating system, eBPF enables the collection & in-kernel aggregation of custom metrics and generation of visibility events based on a wide range of possible sources."

- Cilium, network connectivity security and observability
- Falco, Kubernetes threat detection engine
- Parca, Continuous Profiling
- Tracee, Runtime Security and Forensics

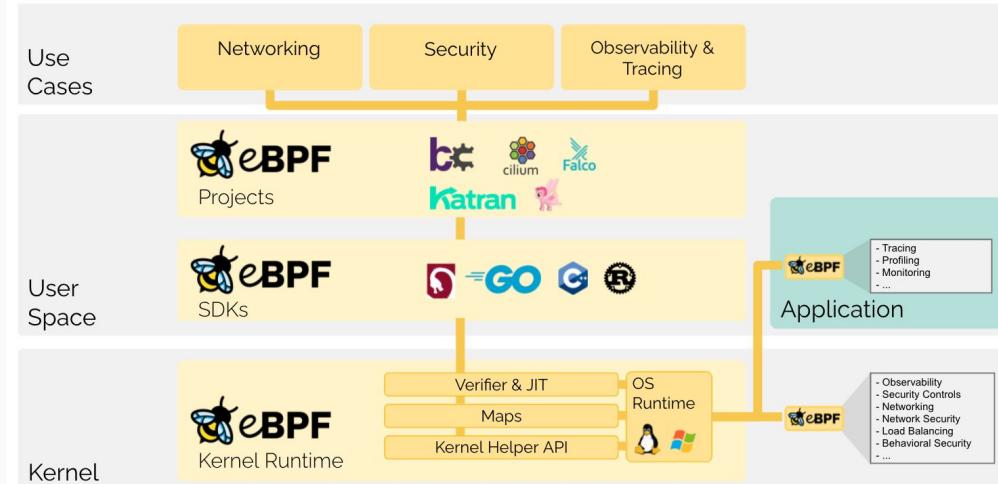
<https://ebpf.io/> - Image credit

<https://cilium.io/>

<https://falco.org/>

<https://www.parca.dev/>

<https://aquasecurity.github.io/tracee/latest/>



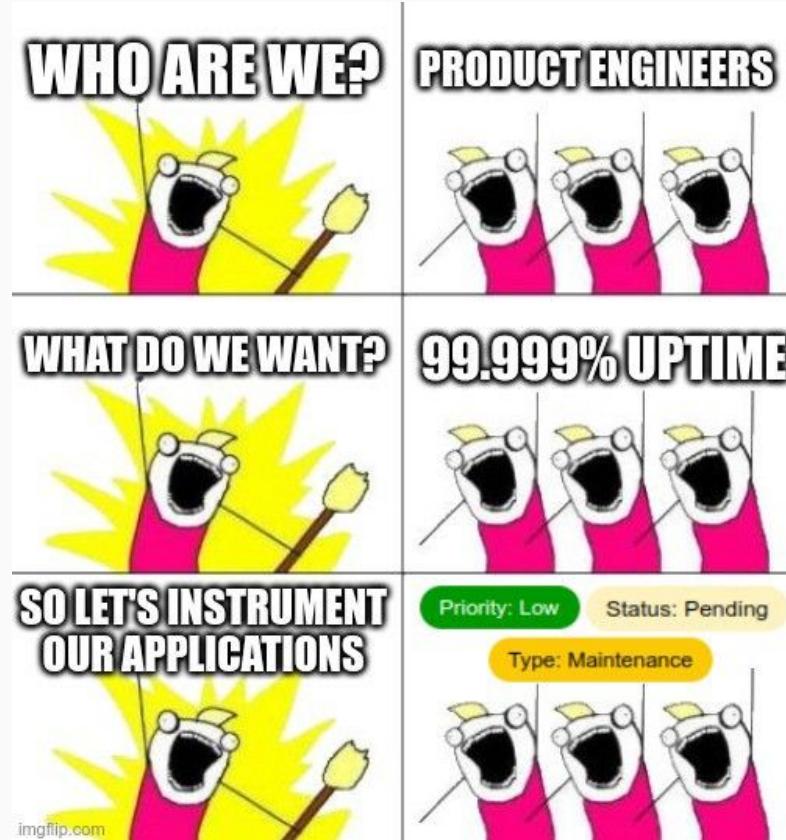
Observability with eBPF

@dnsmichi 

Where does eBPF fit into Kubernetes Observability?

Does it complement / integrate with Prometheus metrics?

Auto-instrumentation for several language VMs (JVM, etc.)



<https://medium.com/@isalapiyaris/getting-started-on-kubernetes-observability-with-ebpf-88139eb13fb2> (image credit)

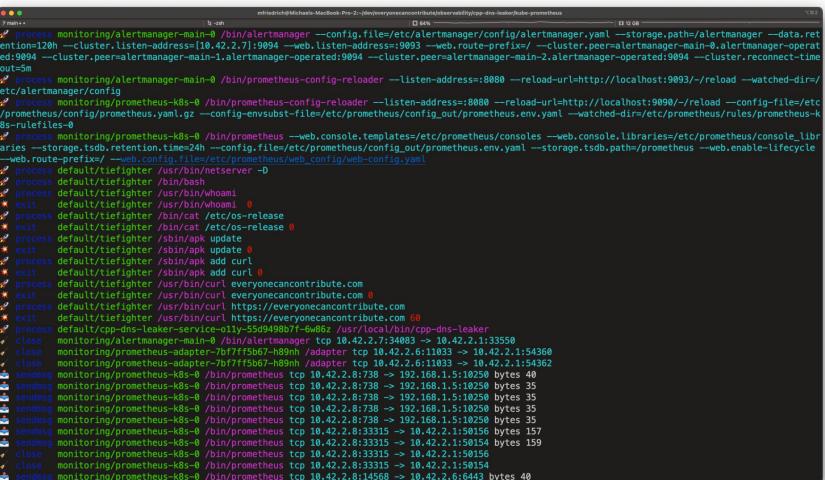
Observability with eBPF

@dnsmichi 

Cilium Tetragon

Pixie

<https://o11y.love/topics/collections-specs/#ebpf>



```
#!/bin/sh
# This script is used to generate a BPF program for monitoring
# and alerting. It uses the libbpf library to define the BPF
# program and register it with the kernel.

# Define the BPF program
cat > /tmp/bpf.ko << EOF
// BPF program code here
EOF

# Build the BPF program
make -C /lib/modules/`uname -r`/build M=/tmp

# Load the BPF program into the kernel
insmod /tmp/bpf.ko

# Print the BPF program
cat /sys/kernel/debug/tracing/bpf/0/program

```

The terminal output shows a large amount of BPF program code, likely generated by a tool like BCC (BPF Compiler Collection). The code defines several BPF programs, each with multiple sections (sections, sections, sections, sections) and various handlers (process, class, sendmsg, receive, etc.) for different kernel events. The events include monitoring/alertmanager, monitoring/prometheus, default/tiefighter, and monitoring/prometheus-k8s. The handlers involve complex logic using functions like bpf_map_lookup_elem, bpf_map_update_elem, and bpf_map_delete_elem.

<https://everyonecancontribute.com/post/2022-06-16-cafe-52-learned-at-kubecon-eu-coffee-chat/>
<https://everyonecancontribute.com/post/2022-09-13-cafe-54-pixie-for-kubernetes-observability/>

EXERCISE: Install Tetragon and observe events

@dnsmichi 

TODO - Advanced topic

```
helm repo add cilium https://helm.cilium.io
helm repo update
helm install tetragon cilium/tetragon -n kube-system
kubectl rollout status -n kube-system ds/tetragon -w

kubectl create -f
https://raw.githubusercontent.com/cilium/cilium/v1.11/examples/minikube/http-sw-app.yaml
```

<https://github.com/cilium/tetragon#deploy-tetragon>

Metrics & Alerts calculation

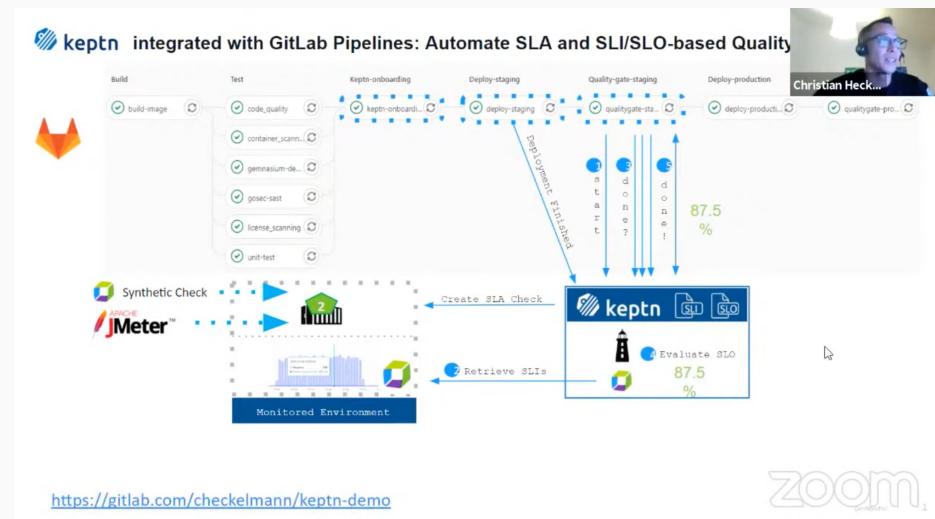
PromQL queries in OpenSLO format

CI/CD

Environments

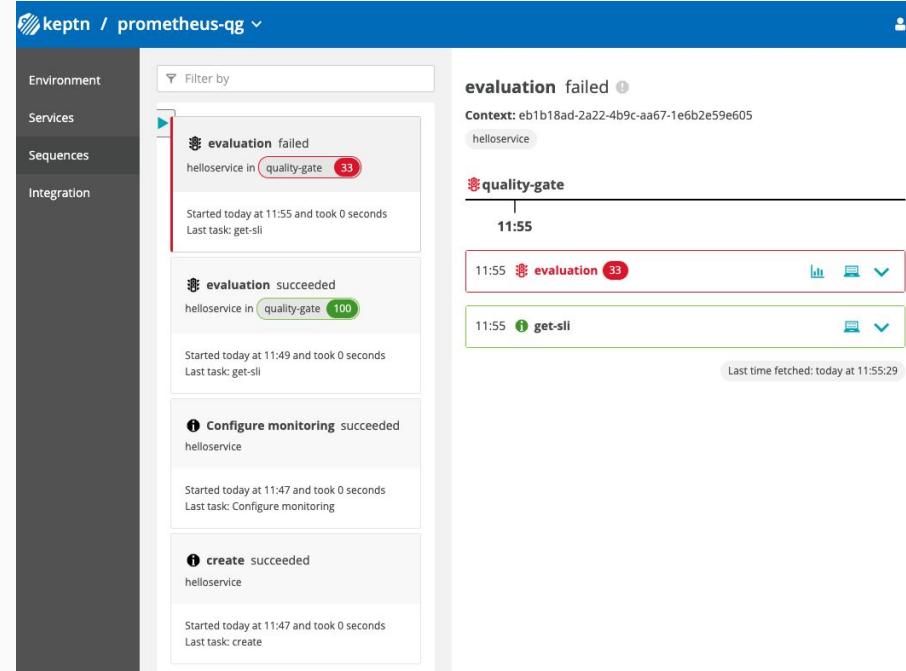
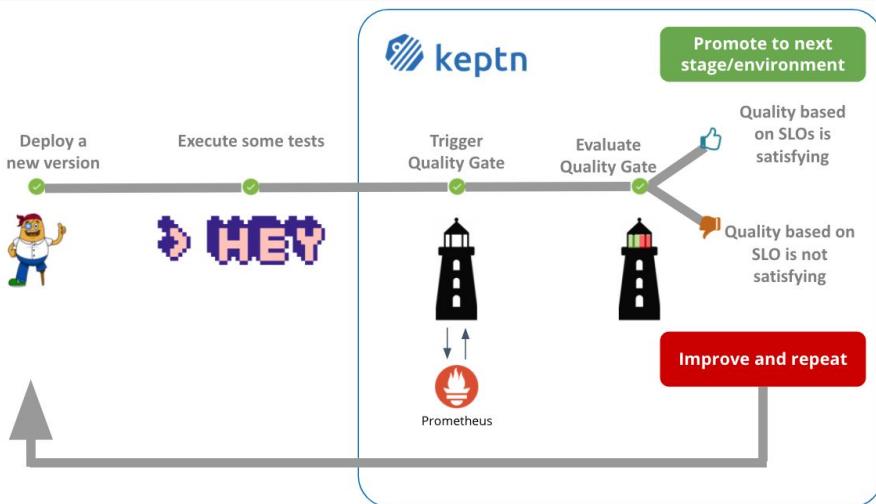
Metrics Monitoring

Keptn with SLOs for quality gates



Oh Keptn, my Keptn

@dnsmichi 



The screenshot shows the Keptn UI interface for the "prometheus-qg" sequence. The left sidebar includes "Environment", "Services", "Sequences", and "Integration" sections. The main area displays a log of events:

- evaluation failed (Context: eb1b18ad-2a22-4b9c-aa67-1e6b2e59e605 helloservice) - Started today at 11:55 and took 0 seconds. Last task: get-sli (33).
- quality-gate (11:55) - evaluation succeeded (Context: eb1b18ad-2a22-4b9c-aa67-1e6b2e59e605 helloservice) - Started today at 11:49 and took 0 seconds. Last task: get-sli (100).
- Configure monitoring succeeded (helloservice) - Started today at 11:47 and took 0 seconds. Last task: Configure monitoring.
- create succeeded (helloservice) - Started today at 11:47 and took 0 seconds. Last task: create.

At the bottom right, it says "Last time fetched: today at 11:55:29".

<https://everyonecancontribute.com/post/2020-11-11-cafe-8-keptn/>

https://keptn.sh/docs/0.9.x/quality_gates/slo/

<https://tutorials.keptn.sh/tutorials/keptn-quality-gates-prometheus-08/#0>

https://www.youtube.com/watch?v=7ksL0V0tN_M

Image credit: [Keptn.sh](https://keptn.sh)

Security



Kubesimplify

Workshops



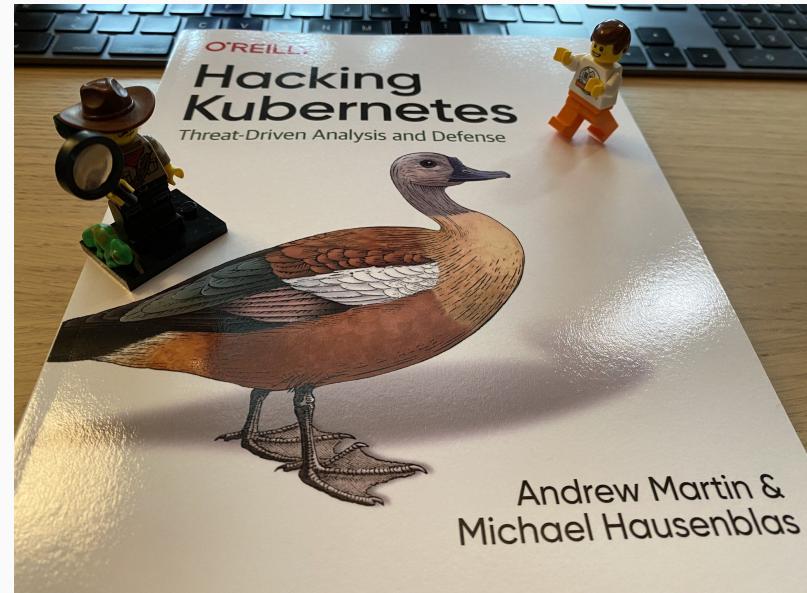
@dnsmichi



- Detect vulnerabilities ***before*** they hurt production
- CI/CD Security Scanning
 - Container images
 - Dependency scanning
 - SAST & DAST
- Kubernetes Security
 - Cluster Image Scanning
 - Open Policy Agent
 - Kyverno policies
- IaC Security Scanning
 - Terraform, Kubernetes, cloud, etc.

<https://about.gitlab.com/blog/2022/02/17/fantastic-infrastructure-as-code-security-attacks-and-how-to-find-them/>
<https://everyonecancontribute.com/post/2021-08-04-cafe-41-kubernetes-cluster-image-scanning-trivy-starboard/>
<https://everyonecancontribute.com/post/2021-08-11-cafe-42-falco-gitlab-package-hunter/>
<https://everyonecancontribute.com/post/2021-08-18-cafe-43-more-package-dependency-hunting-with-gitlab/>

- Server ↔ Exporter communication
 - TLS, basic auth
- Kubernetes namespaces
- Cluster communication
 - mTLS, RBAC, Policies, etc.
 - Book recommendation: “Hacking Kubernetes”



<https://prometheus.io/docs/guides/tls-encryption/>

<https://twitter.com/dnsmichi/status/1503503642158055434>

<https://www.oreilly.com/library/view/hacking-kubernetes/9781492081722/>

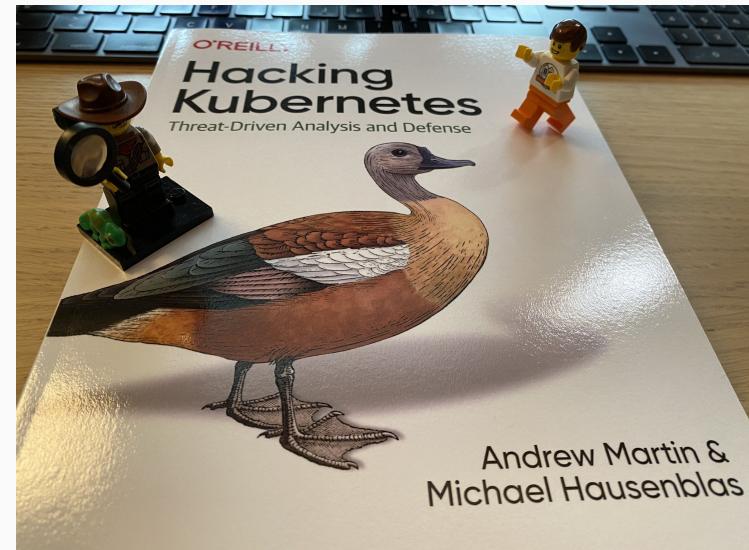
Security Defaults in Kubernetes need improvements

Impersonate the attacker

Use Chaos Engineering principles for pen testing?

Create a Chaos Experiment that downloads
and installs malicious software

Observability for better Security?





Your adventure



Kubesimplify

Workshops



@dnsmichi



Photo by [Michael Friedrich](#)

- Rust
 - https://github.com/prometheus/client_rust
 - <https://rocket.rs/>
 - Book: <https://www.zero2prod.com/>
- Go
- Python
- ...

- Prometheus Metrics
 - <https://o11y.love/topics/metrics/#prometheus>
- OpenTelemetry
 - <https://o11y.love/topics/collections-specs/#opentelemetry>

<https://prometheus.io/docs/practices/instrumentation/>

<https://www.timescale.com/blog/prometheus-vs-opentelemetry-metrics-a-complete-guide/>

<https://o11y.love/learning-resources/>

Q&A



Learn more: <https://o11y.love/>

Subscribe to my newsletter:
<https://opsindev.news/> 

- Prometheus: Up & Running:
<https://www.oreilly.com/library/view/prometheus-up/9781492034131/>
- PromLabs Trainings: <https://training.promlabs.com/>
- Robust Perception Blog:
<https://www.robustperception.io/blog>
- 100 Days of Kubernetes:
<https://100daysofkubernetes.io/observability/prometheus-exporter.html>
- #EveryoneCanContribute cafe:
<https://everyonecancontribute.com/post/2021-05-19-cafe-30-kubernetes-monitoring-prometheus/>
- The exercises in this slide deck 
- ZSH Terminal Theme:
<https://dnsmichi.at/2022/03/11/new-zsh-theme-on-macos-powerlevel10k/>
- My talks: <https://dnsmichi.at/talks/>



Kubesimplify

: Workshops :



@dnsmichi