Kubernetes Observability

Kubesimplify | Workshops

Michael Friedrich
Senior Developer Evangelist at GitLab
@dnsmichi

EveryoneCanContribute.com

What to expect

Intro and preparations

Monitoring, quo vadis

Prometheus, Grafana, Kubernetes

Alerts, incidents, SLOs

Observability?

Tracing, logs, and more

Scaling, eBPF, security, ideas

Workshop Requirements

- 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>
```

<helpful URLs here>

Workshop Projects

- Exercises and solutions
  - https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-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-o11y-2022.git
```

<Help URLs here>

Kubernetes Cluster

- 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

Kubernetes Cluster: Civo Cloud

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

```
$ civo kubernetes create ks-k8s-o11y

$ civo kubernetes ls
$ civo kubernetes show ks-k8s-o11y

$ civo kubernetes config ks-k8s-o11y --save
--merge

$ kubectl config use-context ks-k8s-o11y

$ kubectl get node
```

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

Tips

- 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

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Monitoring in a nutshell

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

Blackbox Monitoring

“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?

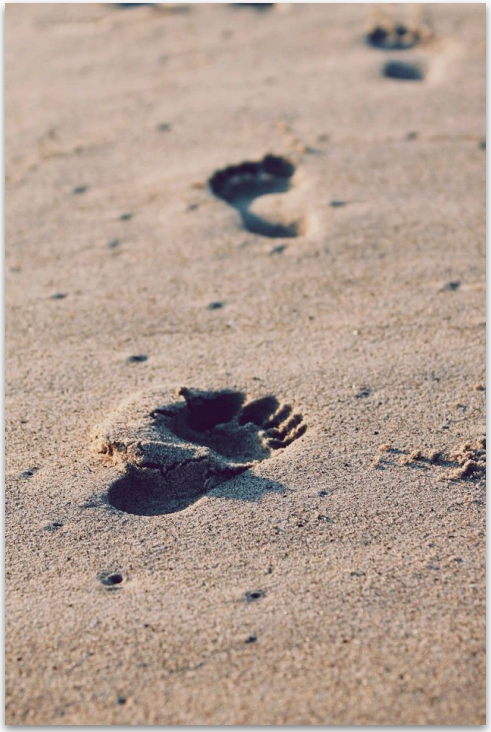


Photo by Christopher Sardegna on Unsplash

Log Insights

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

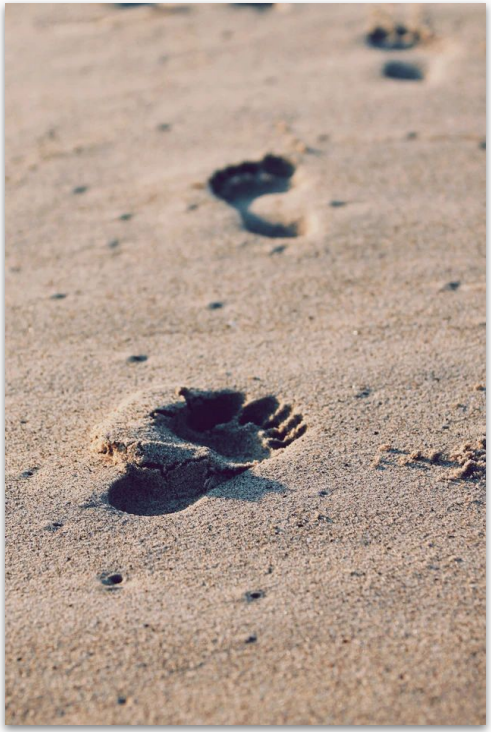


Photo by Christopher Sardegna on Unsplash

Metrics and Trends

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?

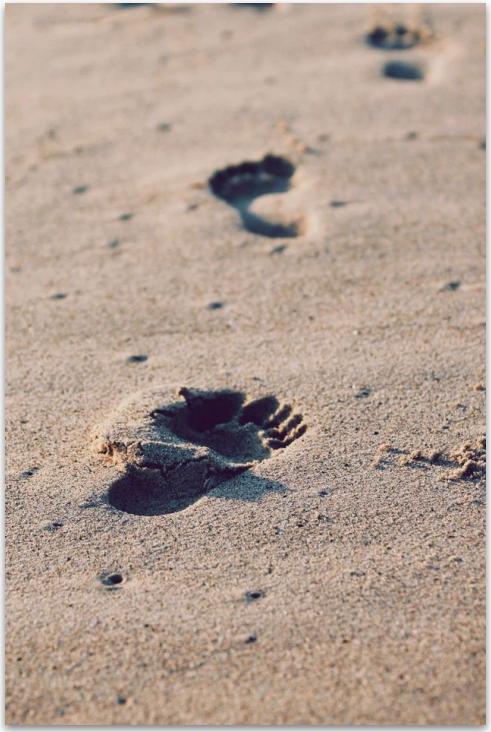


Photo by Christopher Sardegna on Unsplash

Microservices and classic host/service models

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

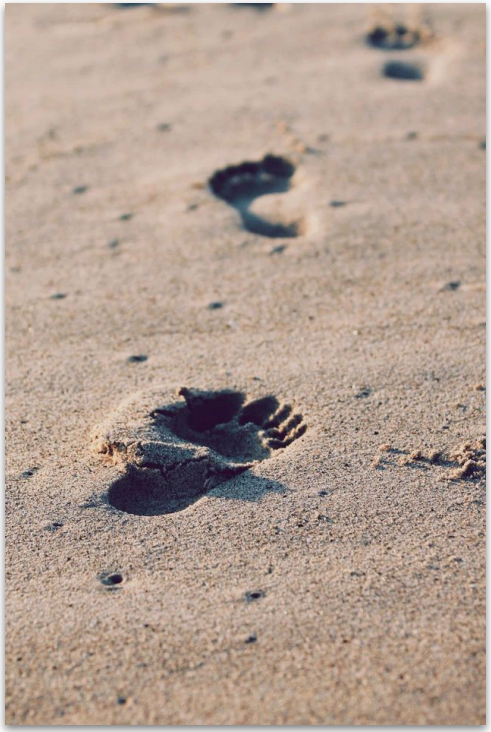


Photo by Christopher Sardegna on Unsplash

Whitebox Monitoring

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

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

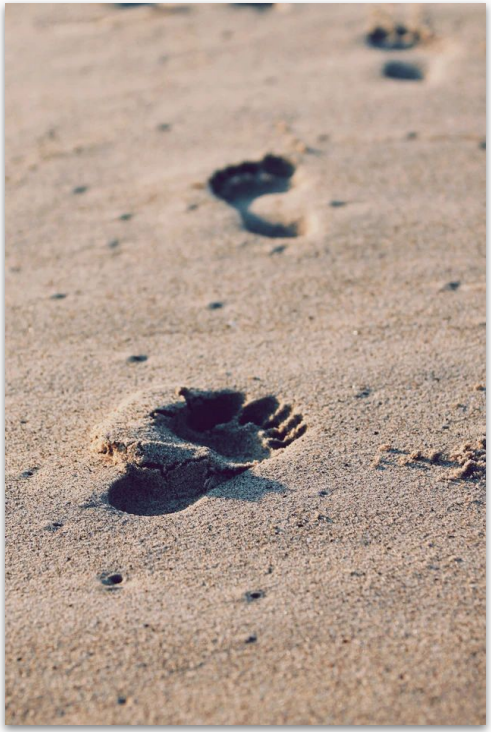


Photo by Christopher Sardegna on Unsplash

Kubesimplify | Workshops

@dnsmichi

Photo by Michael Friedrich

Kubernetes up and running

Kubernetes

Architecture, Components

Nodes, Pods, Containers

Deployments, Services, …

APIs, ports, data sources

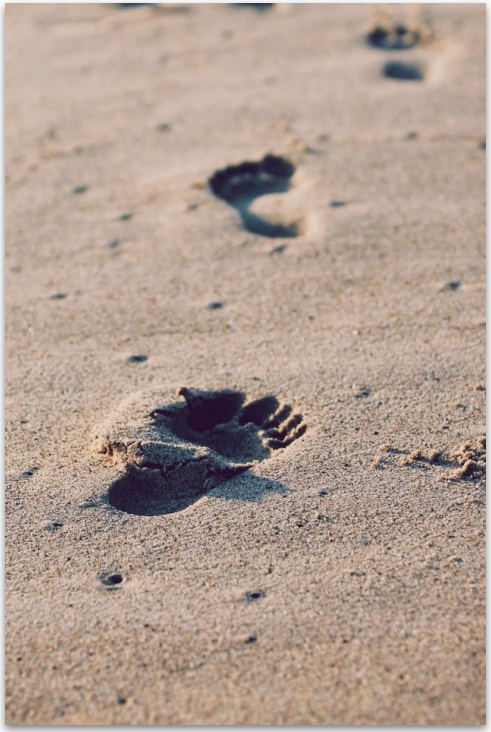


Photo by Christopher Sardegna on Unsplash

We need monitoring

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

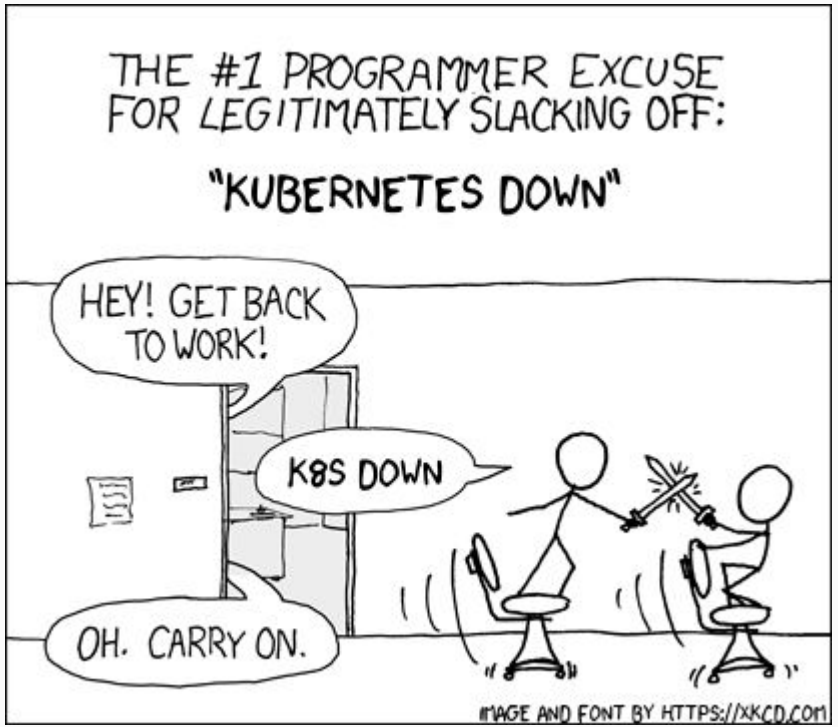


Image generated with https://gitlab.com/gitlab-de/go-excusegen which itself is based on the XKCD comic at https://xkcd.com/303/

Thoughts and Tips

- 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

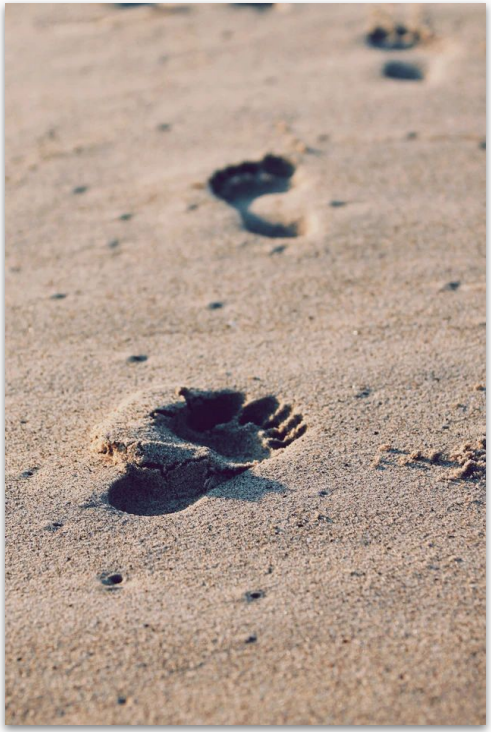


Photo by Christopher Sardegna on Unsplash

EXERCISE: Inspect Kubernetes and learn what to monitor

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
```

https://kubernetes.io/docs/tasks/debug-application-cluster/

Metrics with Prometheus

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Monitoring, first iteration: Metrics

Kubernetes

Different data sources

Service discovery?

CNCF ecosystem

Prometheus

/metrics

TSDB, trends, dashboards

SLOs

Alerts & incidents

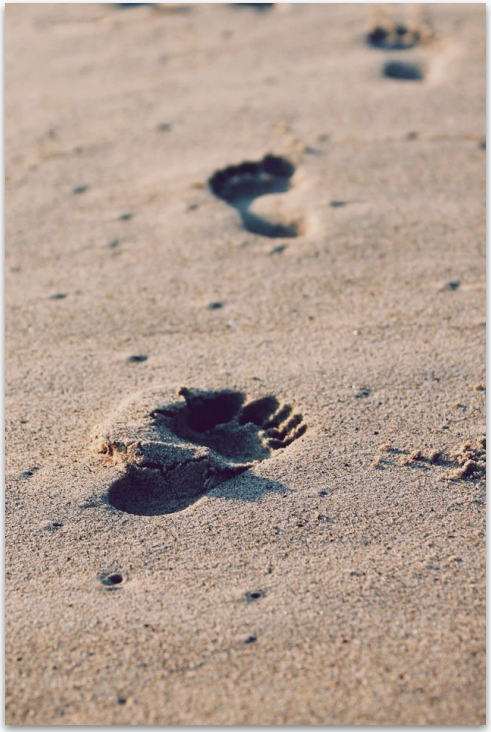
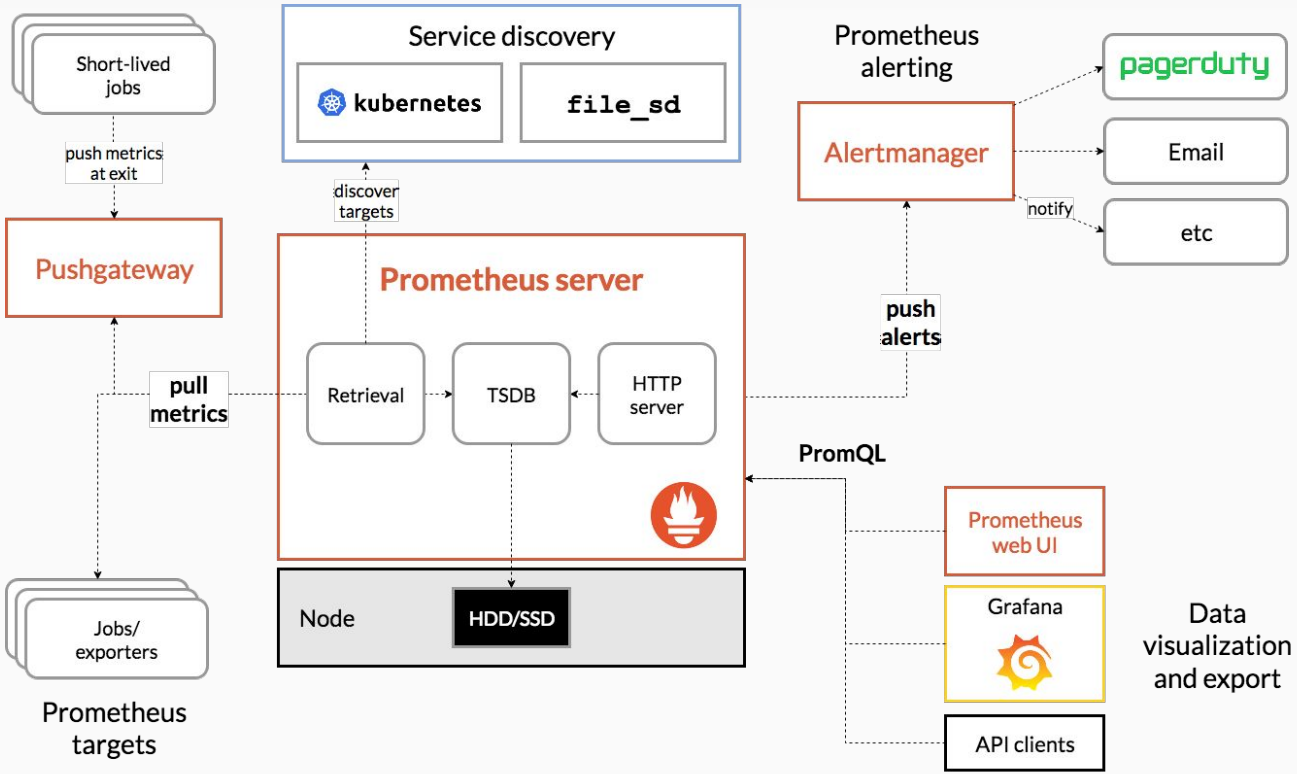


Photo by Christopher Sardegna on Unsplash

Prometheus

- 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



Picture from https://prometheus.io/docs/introduction/overview/#architecture

Prometheus: PromQL

“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**.”

```
# Latest sample
metric_name

# Range
metric_name[5m]

# Labels
metric_name{label1=”a”,label
2=”b”}

# Functions
rate(metric_name[5m])
sum(metric_name)
delta(metric_name[5m])

# Comparisons
metric_name > 10*1024
```

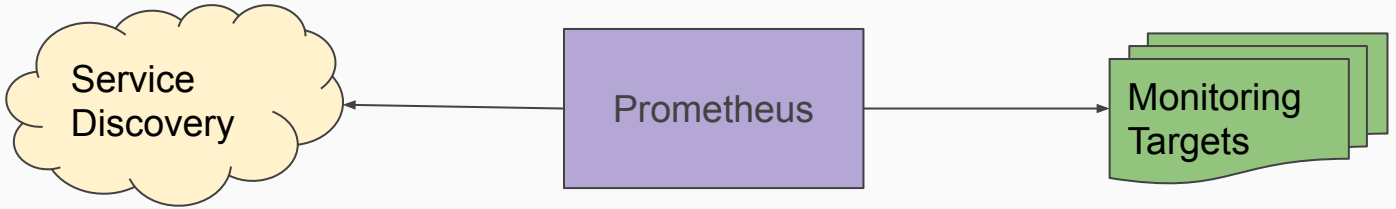
https://prometheus.io/docs/prometheus/latest/querying/examples/
https://promlabs.com/promql-cheat-sheet/
https://demo.promlens.com/

Prometheus: UI

- 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

Prometheus Terminology: Service Discovery

- 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



https://prometheus.io/docs/guides/file-sd/

Monitoring Server (Prometheus) inside Kubernetes?

- 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

Introduction: Prometheus Operator

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.

https://prometheus-operator.dev/docs/prologue/introduction/ - slide content copied from the docs.

kube-prometheus

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

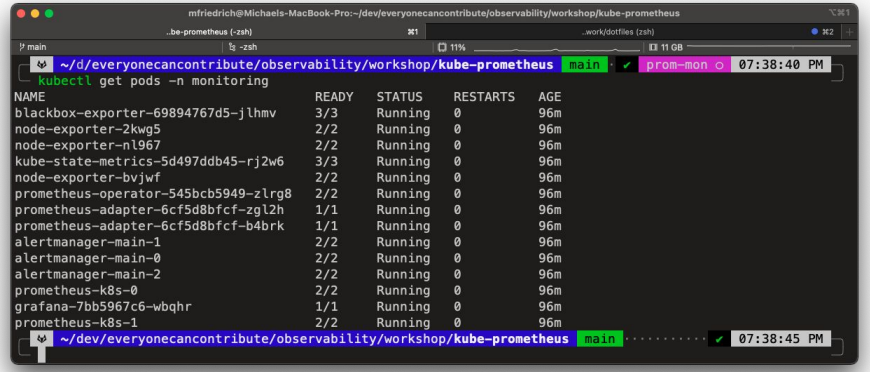
EXERCISE: Install kube-prometheus & Prometheus Operator

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-o11y-2022.git  && cd
k8s-o11y-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

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.




```
$ 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 &
```

EXERCISE: Install NGINX ingress controller

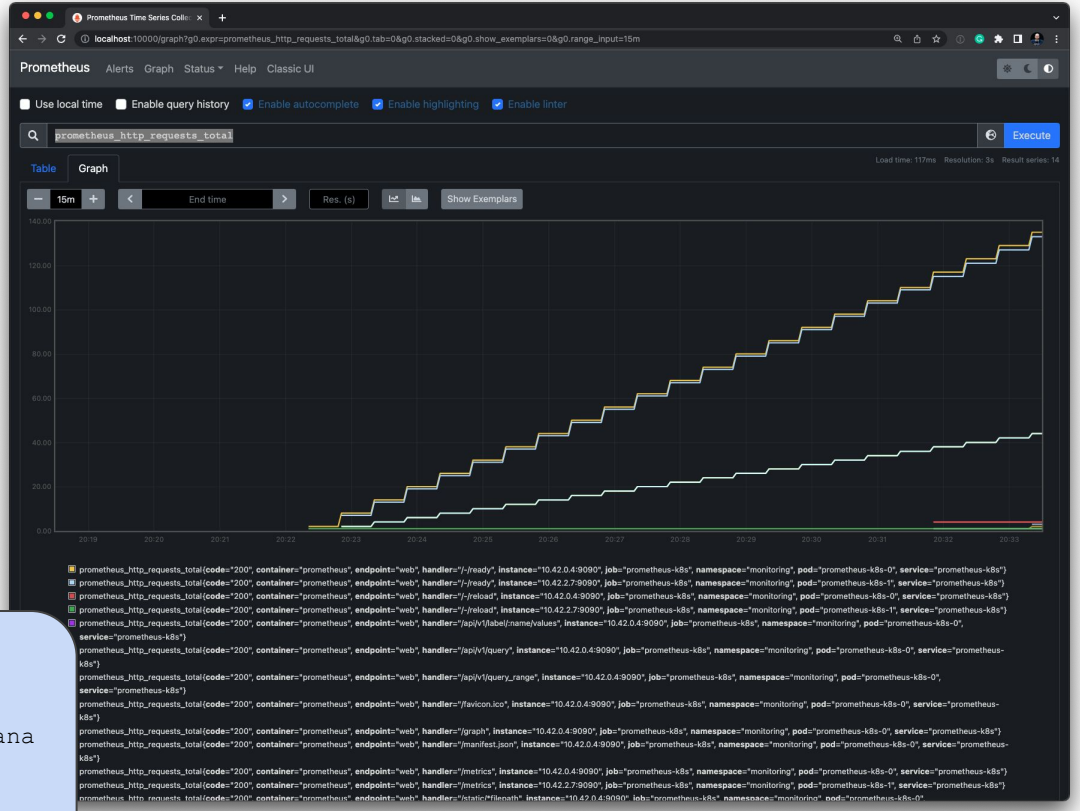
@dnsmichi

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

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

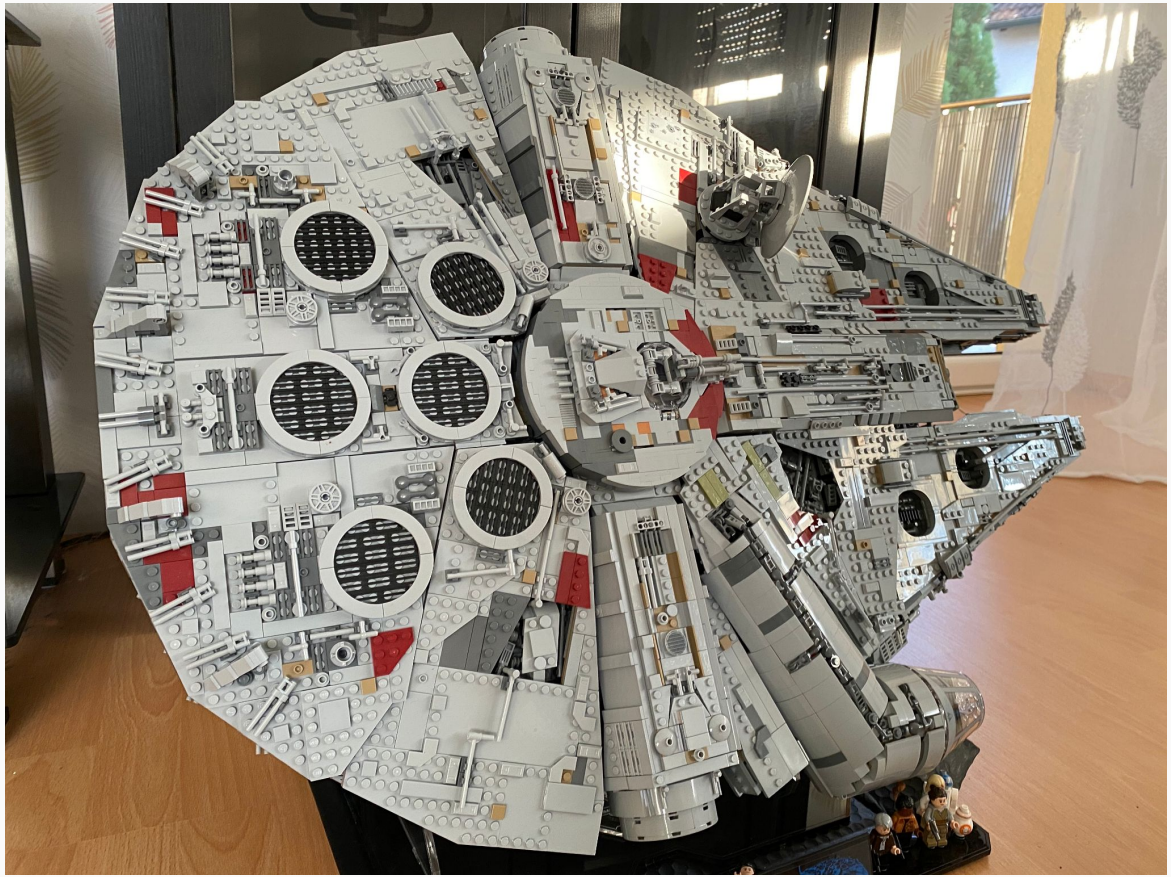


Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Metrics?

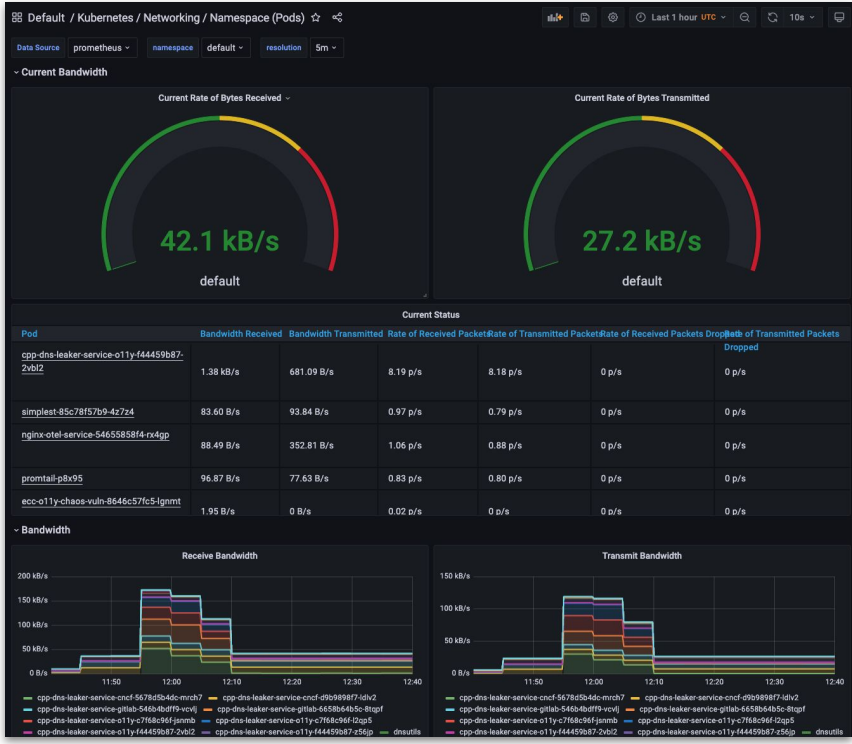
Cluster metrics

- Node status, resource usage
- Deployments, no of pods

Pod metrics

- Kubernetes, containers, applications

So many metrics …



Container 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

https://github.com/google/cadvisor
https://kubernetes.io/docs/concepts/cluster-administration/system-metrics/#metrics-in-kubernetes

kube-state-metrics

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

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Grafana Introduction

“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

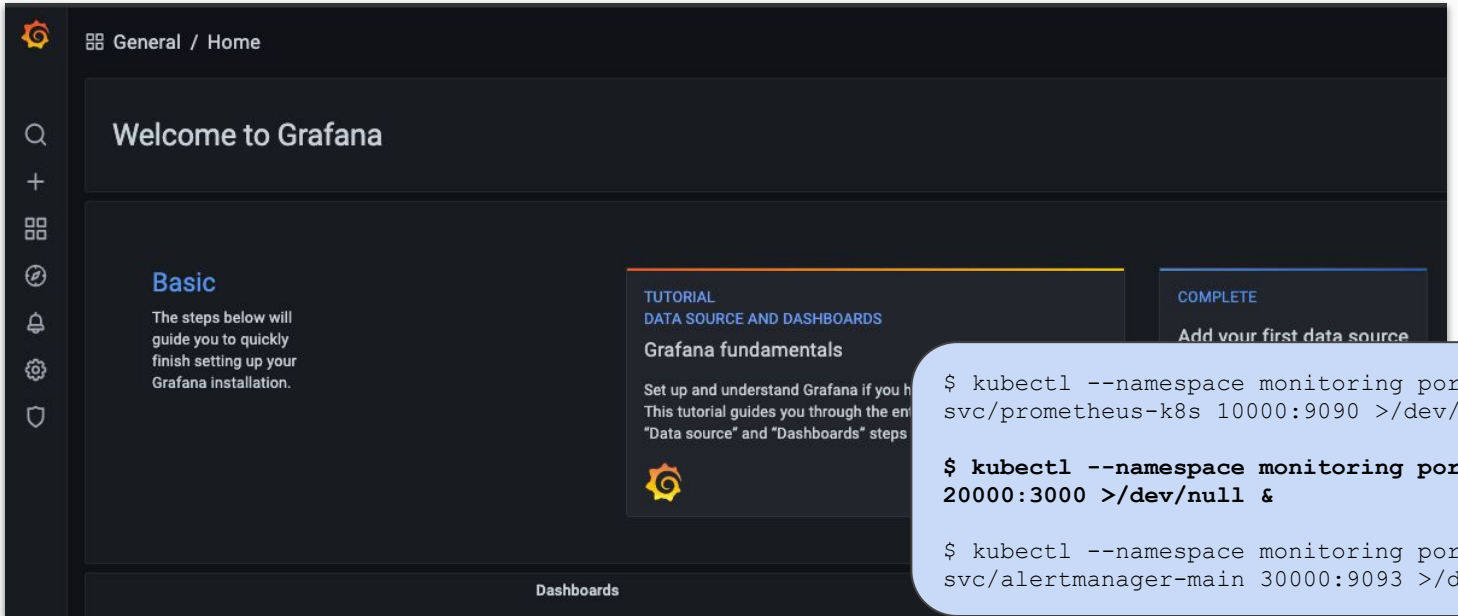
Grafana Terminology: Dashboards, Panels, Data sources

- 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

EXERCISE: Verify

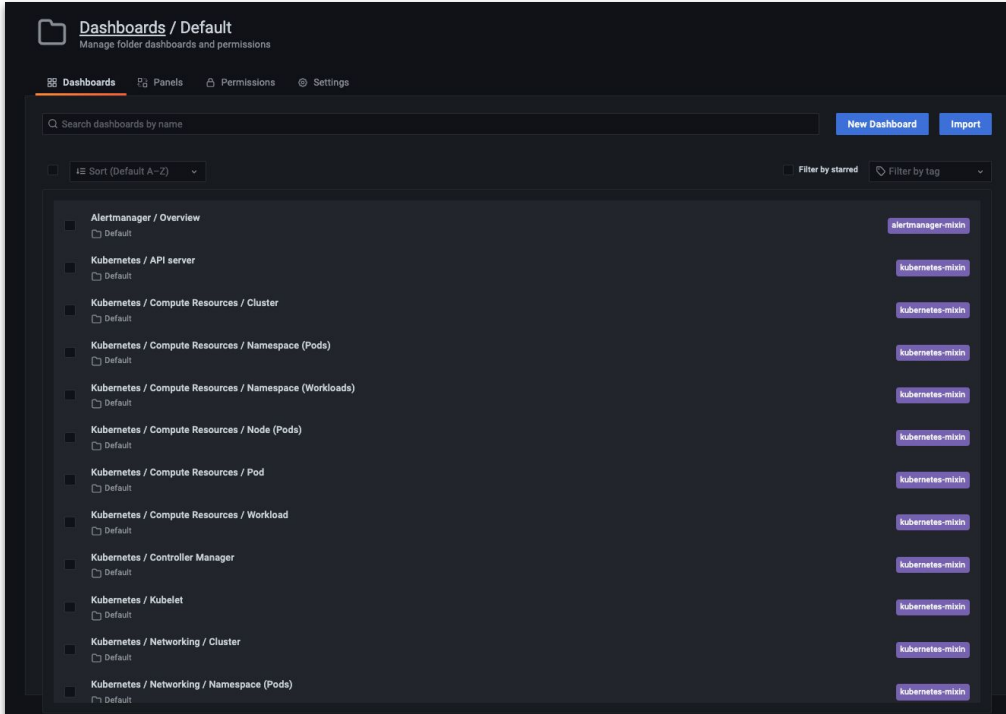
1. Open http://localhost:20000 to access Grafana UI
    a. Grafana requires `admin:admin`, skip changing the password.




```
$ 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 &
```

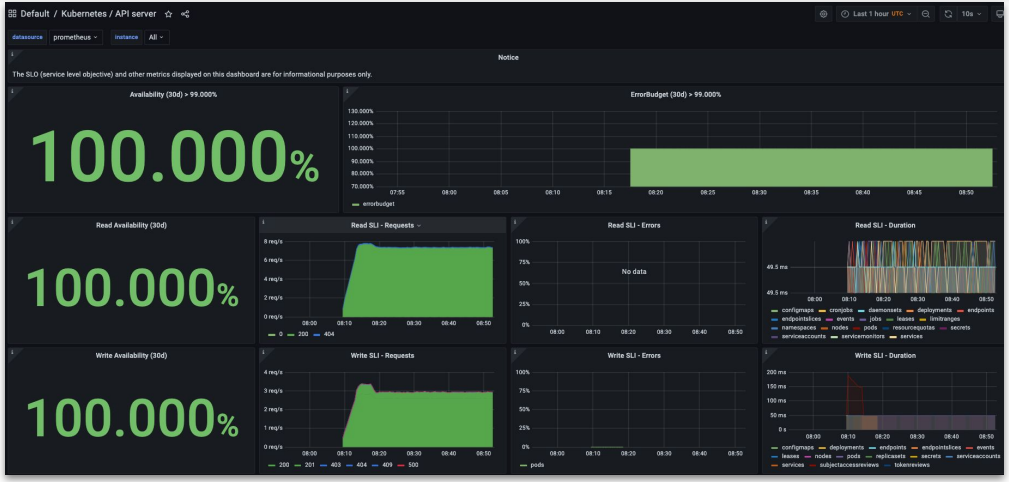
EXERCISE: Grafana dashboards

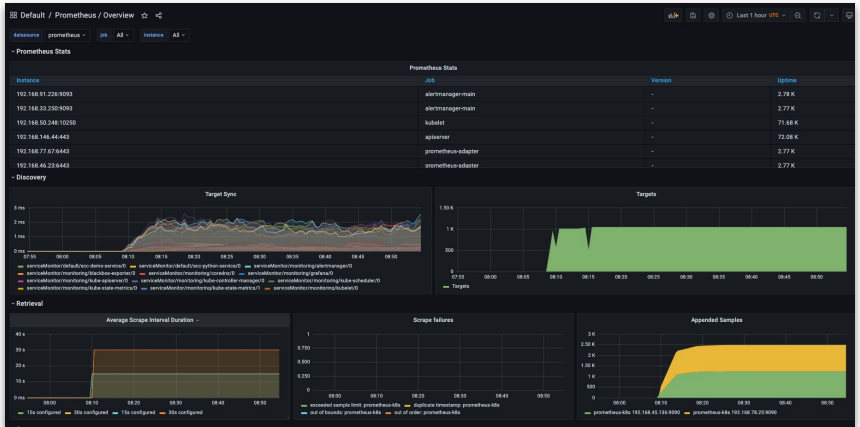
1. Open Grafana at http://localhost:20000
2. Navigate to `Dashboards > Manage > Default` and inspect them
   a. Kubernetes API Server
   b. Compute Resources / Cluster
   c. Kubelet
   d. Prometheus Overview

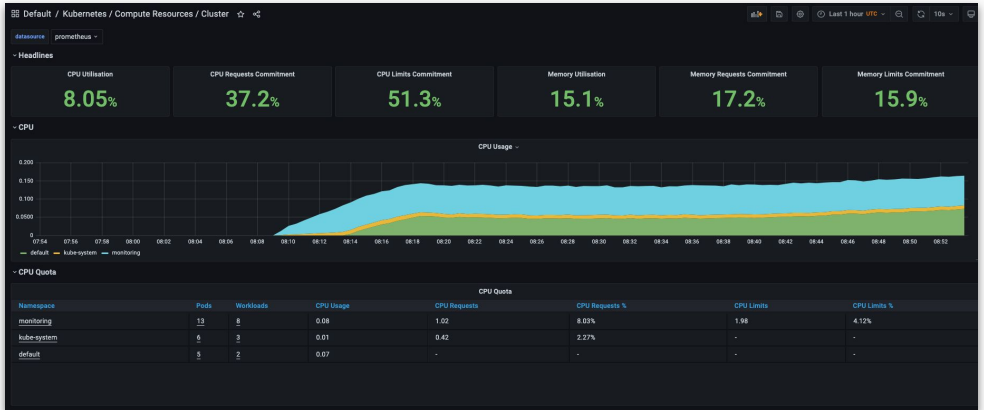


Grafana dashboards

@dnsmichi



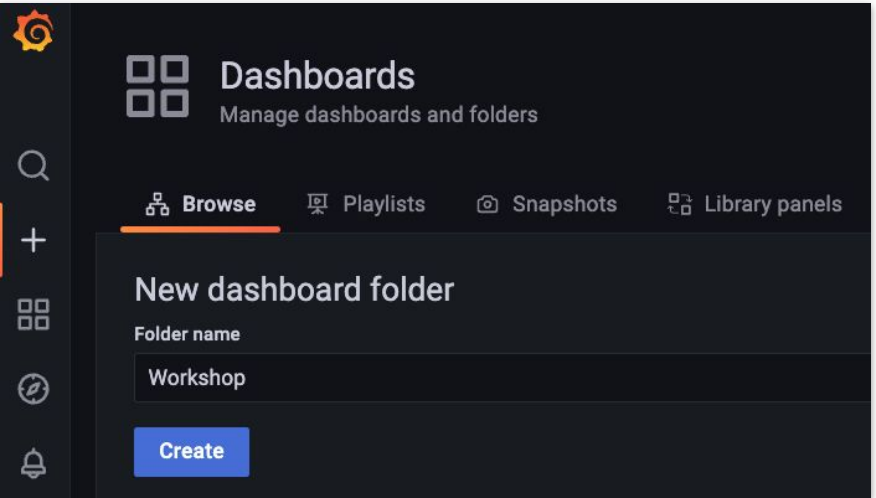


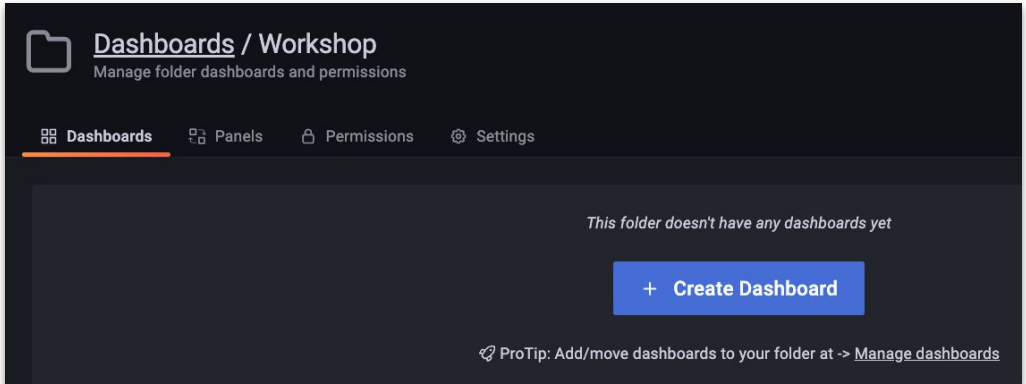


EXERCISE: Workshop dashboards in Grafana

@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`
4.

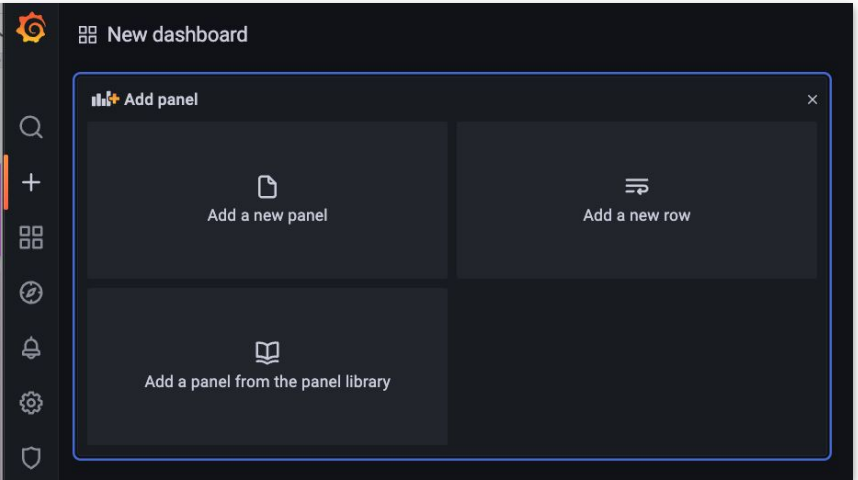


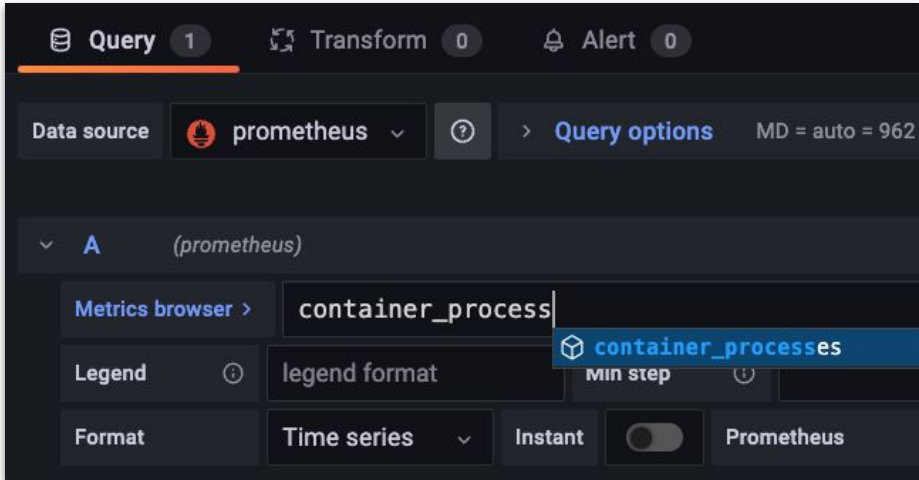


EXERCISE: First panel with PromQL and Grafana

@dnsmichi

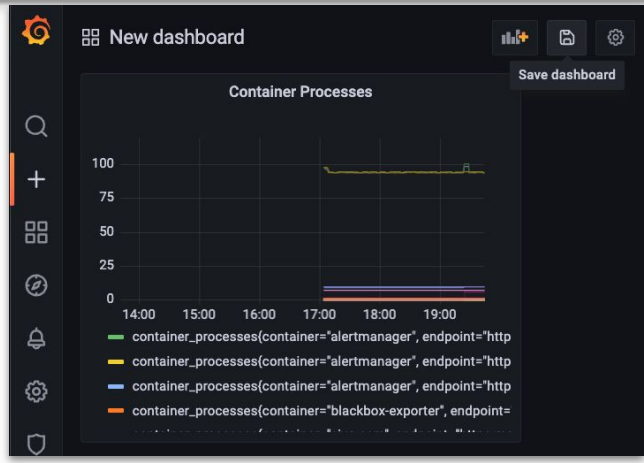
1. Create the new dashboard
2. Add a new panel
3. Data source: **Prometheus**
4. Metrics browser: **container_processes**

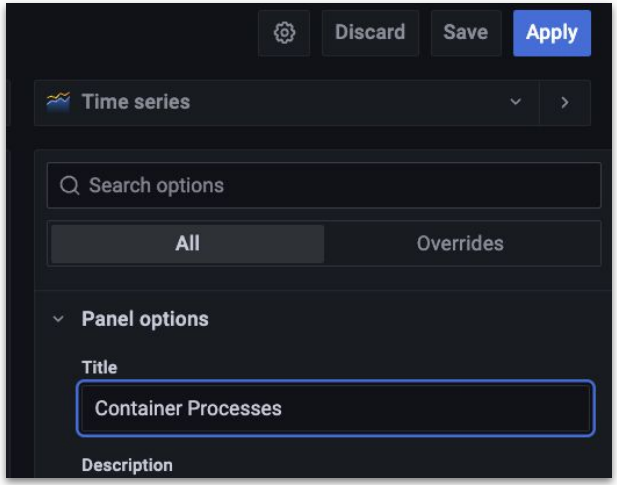


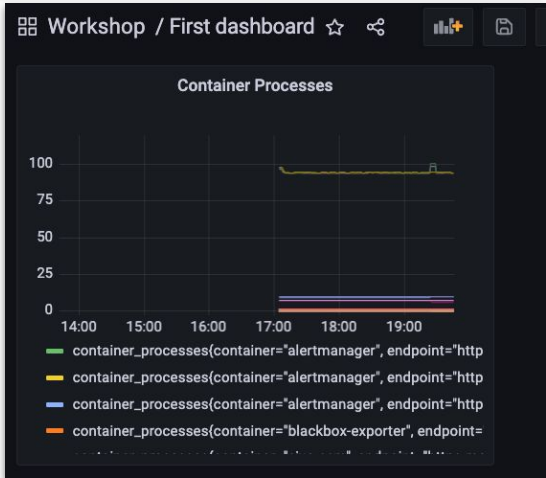


EXERCISE: Save the panel

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



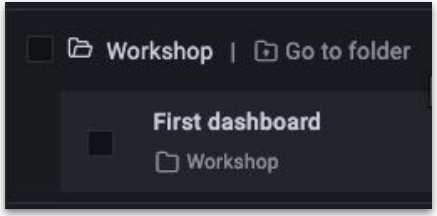


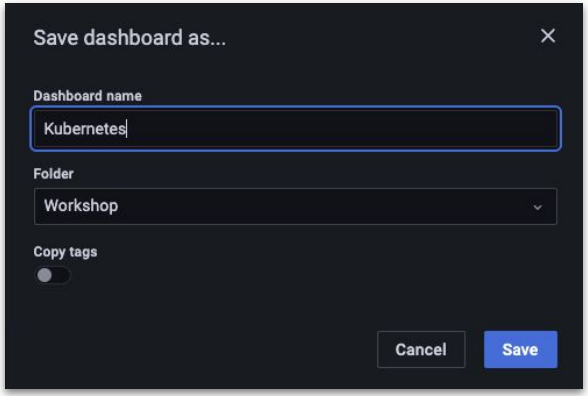


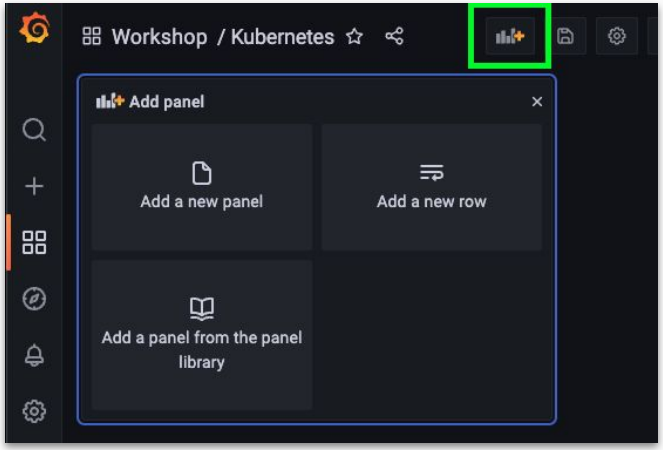
EXERCISE: Prepare Kubernetes Dashboard

@dnsmichi

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







Reminder: Container Metrics

- 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

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

EXERCISE: Query Container Metrics with PromQL

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"}
```

Reminder: kube-state-metrics

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

EXERCISE: Query kube-state-metrics with PromQL

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"}
```

Monitor deployed applications with /metrics endpoint

@dnsmichi

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

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

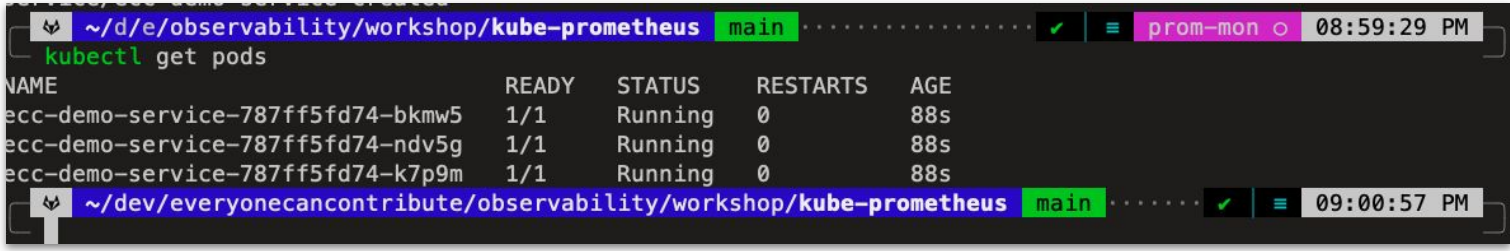
```
$ kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/prometheus-demo-service/ecc-demo-service.yml
```

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

- Verify the created pods (3)

```
$ kubectl get pods
```



EXERCISE: Monitor the Prometheus Demo Service

- 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

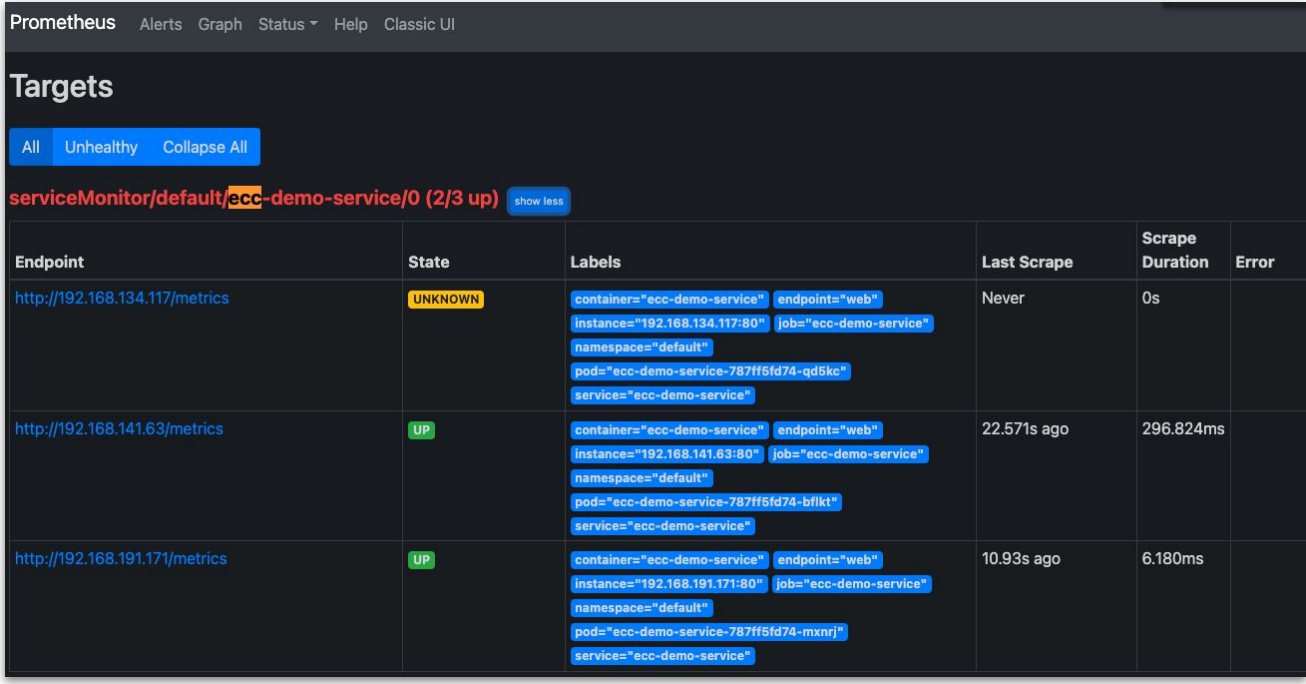
```
$ kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/
prometheus-demo-service/ecc-demo-service-monitor.yml

$ 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

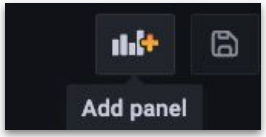
@dnsmichi

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

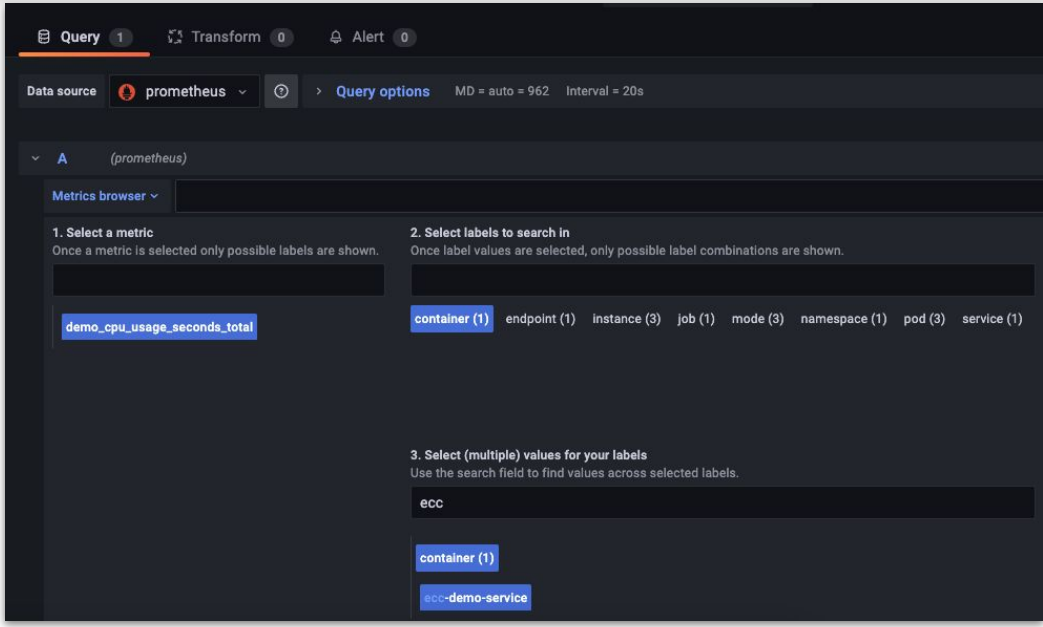


EXERCISE: Query demo service metrics in Grafana

@dnsmichi



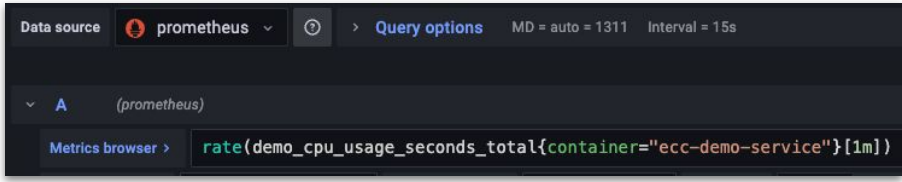
1. Add panel
2. Query
3. Data source: Prometheus
4. Metrics Browser:
   a. Search for demo_cpu
   b. **demo_cpu_usage_seconds_total**
   c. Label: **container**
   d. Values: **ecc-demo-service**
5. Use Query




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

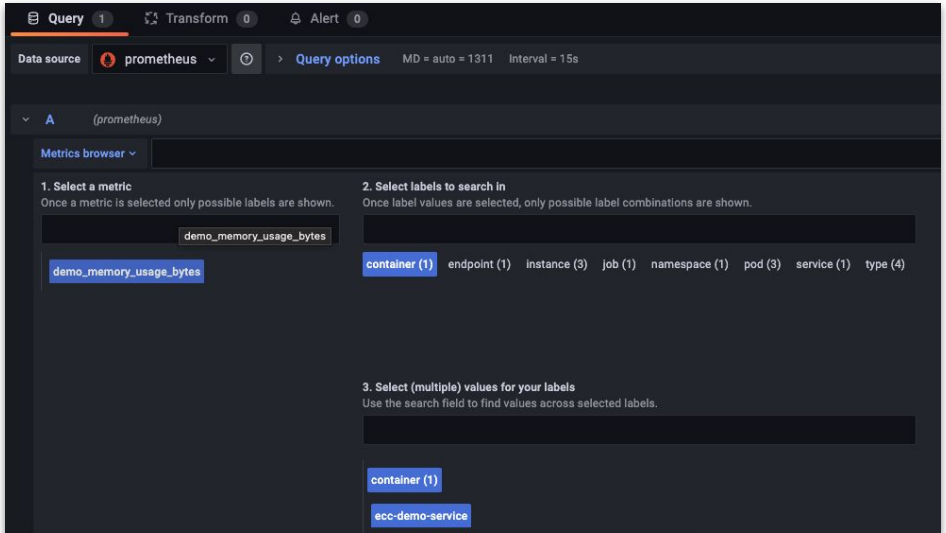
Modify the query to use the rate as suggested

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



EXERCISE: Query demo service metrics in Grafana - practice

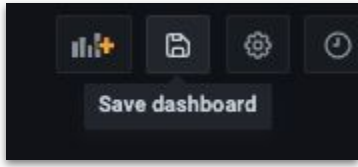
1. Apply the first panel
   a. Title: CPU Usage Seconds
2. Create a new panel
3. Add empty panel > Query
   a. Datasource: Prometheus
4. Search for **demo**
5. Build a new query
6. Use query

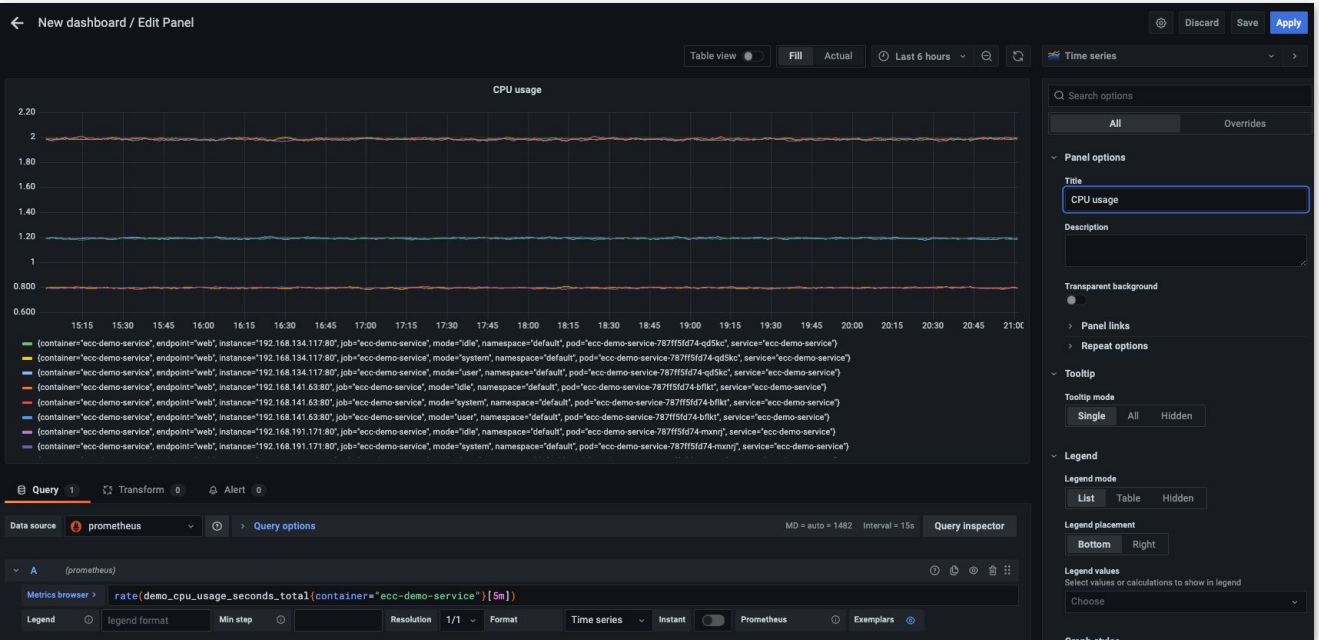


EXERCISE: Query demo service metrics in Grafana - practice

@dnsmichi

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





Monitoring 2.0

App instrumentation

Customizing

Kubesimplify | Workshops

@dnsmichi

As a Developer, where to?

SLO

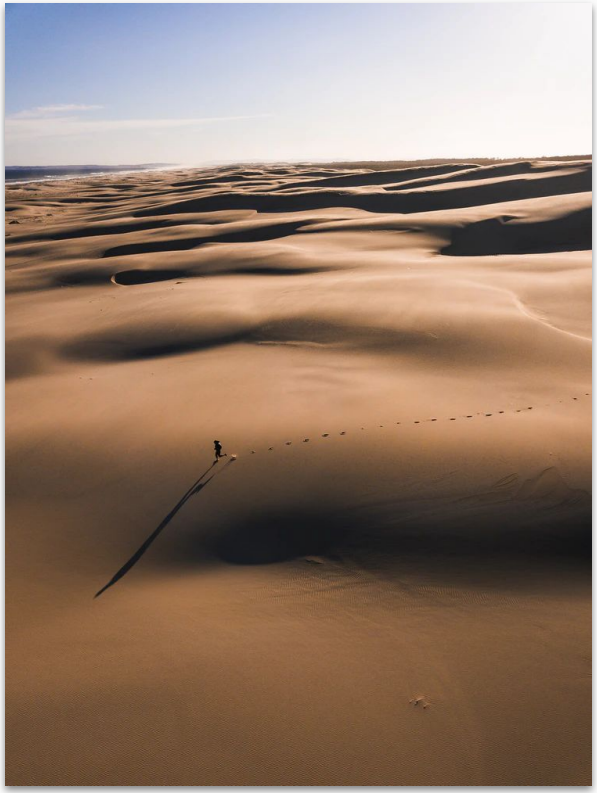
Monitoring

Metrics

key/tag

Values

?



EXERCISE (advanced): Practice - Python app prometheus_python_service

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

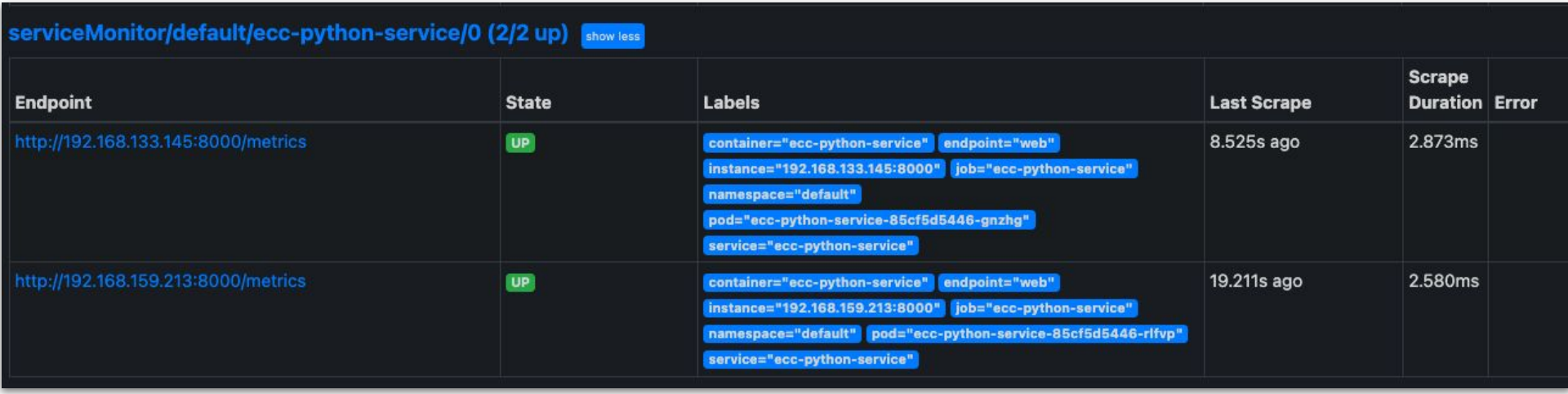
```
$ kubectl create -f
https://gitlab.com/everyonecancontribute/observability/prometheus_python_service/-/raw/main/manifests/ecc-python-service.yml

$ 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

EXERCISE (advanced): Grafana dashboards for prometheus_python_service

@dnsmichi

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	container="ecc-python-service" endpoint="web" instance="192.168.133.145:8000" job="ecc-python-service" namespace="default" pod="ecc-python-service-85cf5d5446-gnzhg" service="ecc-python-service"	8.525s ago	2.873ms	
http://192.168.159.213:8000/metrics	UP	container="ecc-python-service" endpoint="web" instance="192.168.159.213:8000" job="ecc-python-service" namespace="default" pod="ecc-python-service-85cf5d5446-rlfvp" service="ecc-python-service"	19.211s ago	2.580ms	

https://github.com/prometheus/client_python#three-step-demo
https://gitlab.com/everyonecancontribute/observability/prometheus_python_service

EXERCISE (Advanced): Changes for 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

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Metrics are great - what's next?

(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

Alerts with Prometheus

- 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/

Alerts for Kubernetes

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
```

https://awesome-prometheus-alerts.grep.to/rules.html#kubernetes
https://www.zupzup.org/prometheus-alert-examples/

Alert Rules in Prometheus Operator

- 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

Alert Receivers

Chat: “See something immediately”

Refinements and grouping needed

- Ticket and issue systems
- Mailing lists

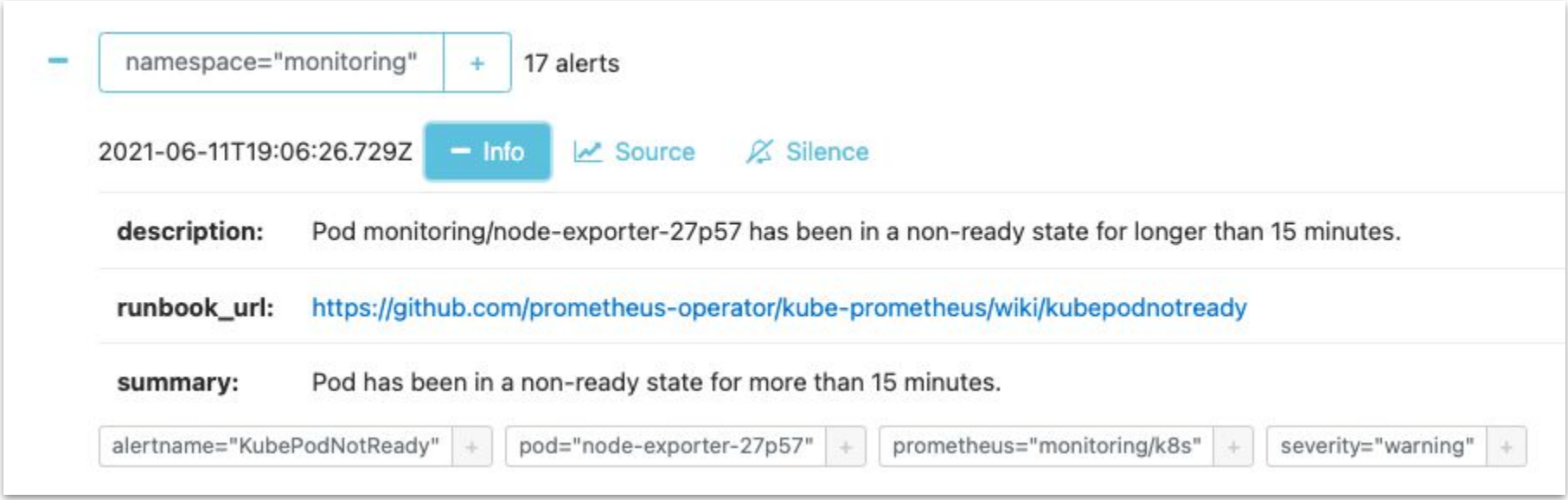
Avoid alert fatigue

Dedicated incident and issue management platforms

https://training.promlabs.com/training/alerting-with-prometheus

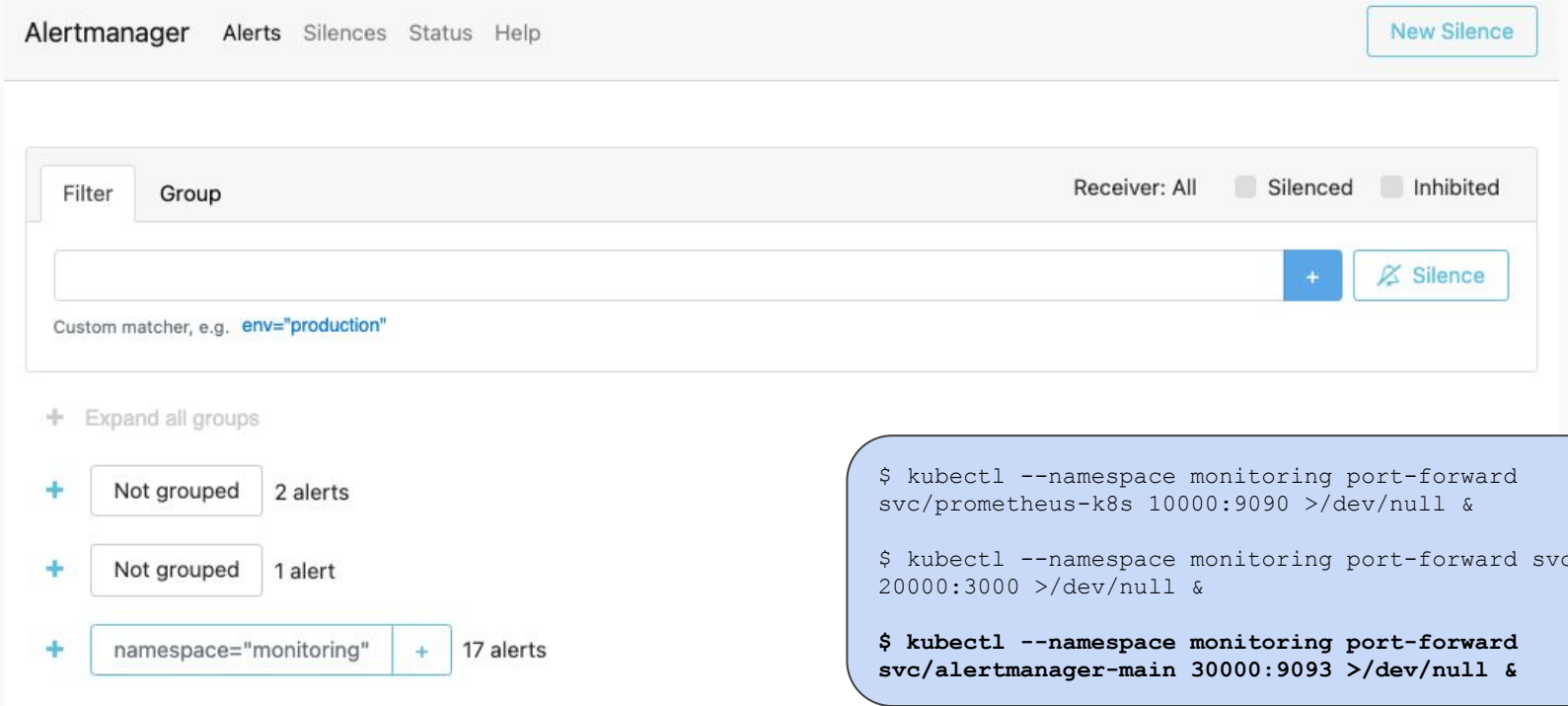
Trigger alerts manually - kill some pods

Manual work is a bug!



EXERCISE: Verify setup with Alert Manager UI

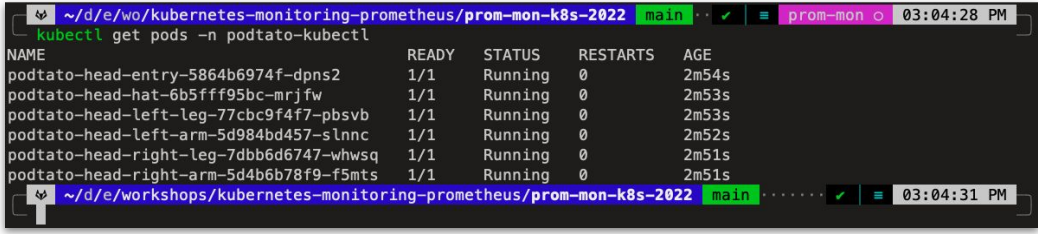
1. Open http://localhost:30000 to access Alert Manager UI




```
$ 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 &
```

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

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



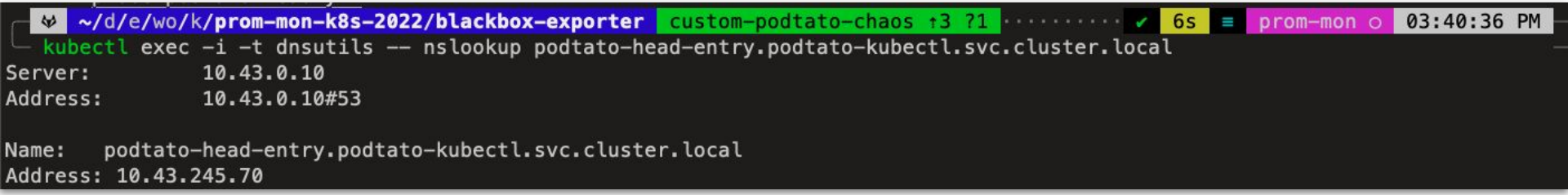

```
$ kubectl create -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/
podtato-head/manifest.yaml

$ 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**

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
```



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

EXERCISE: Trigger alerts 2/x - Monitor podtato-head with blackbox exporter

@dnsmichi

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-o11y-2022/-/raw/main/kubernetes/o11y/podtato-head/probe-podtato-head.yml

$ kubectl get probes -n monitoring
```

https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/podtato-head/probe-podtato-head.yml

```
$ 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 &
```

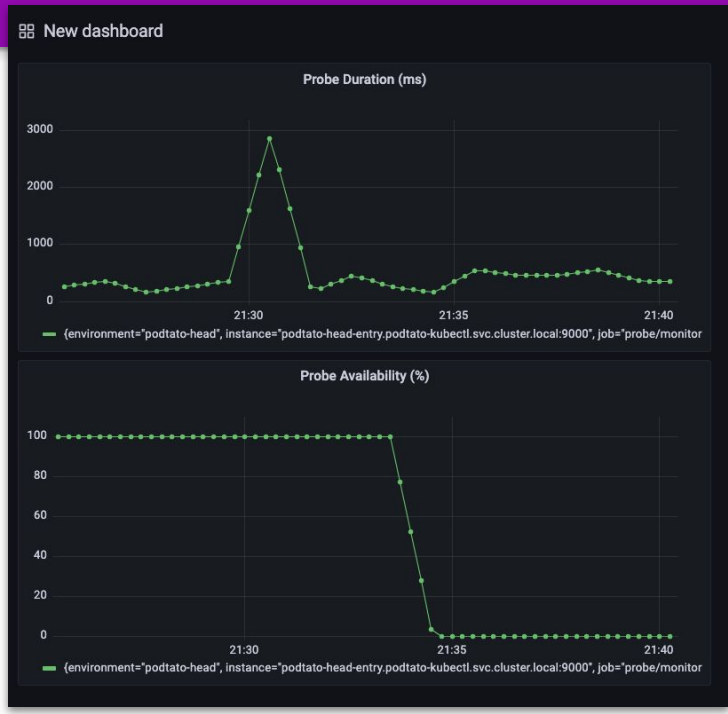
EXERCISE: Trigger alerts 3/x - Add grafana dashboard

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
```



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

```
$ 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 &
```

EXERCISE: Trigger alerts 4/x - Alert rules for podtato-head Probe

1. Add PrometheusRule for alerts

```
$ kubectl apply -f
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-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-o11y-2022/-/blob/main/kubernetes/alerts/podtato-head-probe-alerts.yml

EXERCISE: Trigger alerts 5/x - Delete podtato-head deployment

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
```

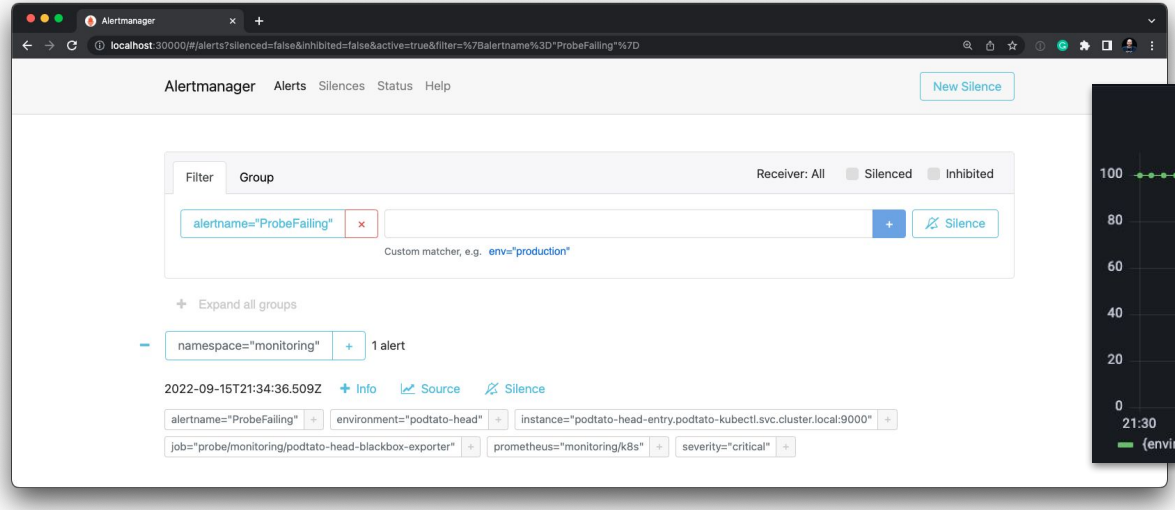
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/raw/main/kubernetes/o11y/podtato-head/manifest.yaml

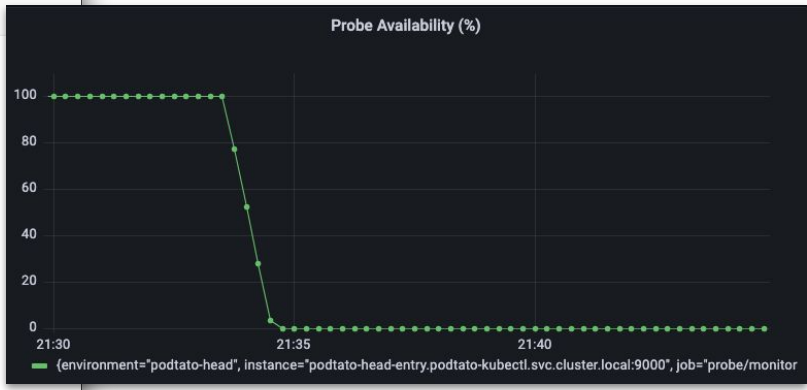
EXERCISE: Trigger alerts 6/x - Verify alerts and graphs

@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 &
```





Moving into SL*

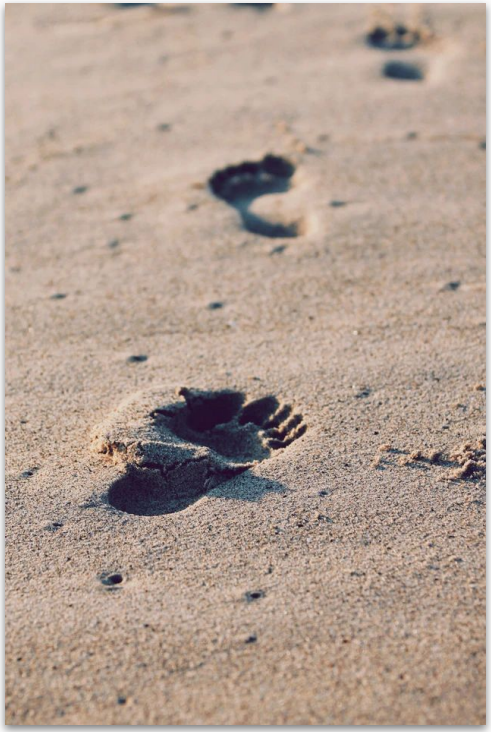
Service Level

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

From metrics to SLOs

Monitoring

Metrics

key/tag

Values

Alerts on PromQL queries

SLOs

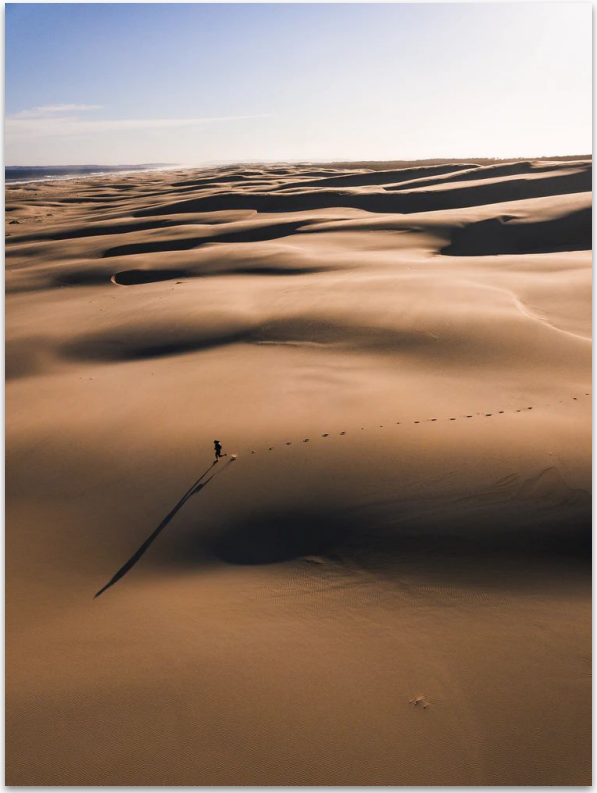


Photo by Silas Baisch on Unsplash

Ops SLOs: Start with Golden Signals

- Latency
- Traffic
- Errors
- Saturation

https://sysdig.com/blog/golden-signals-kubernetes/
https://sre.google/books/

Image credit: Denise Yu

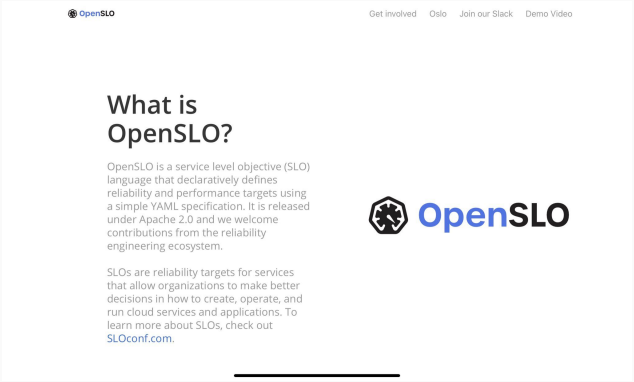
SLOs as code

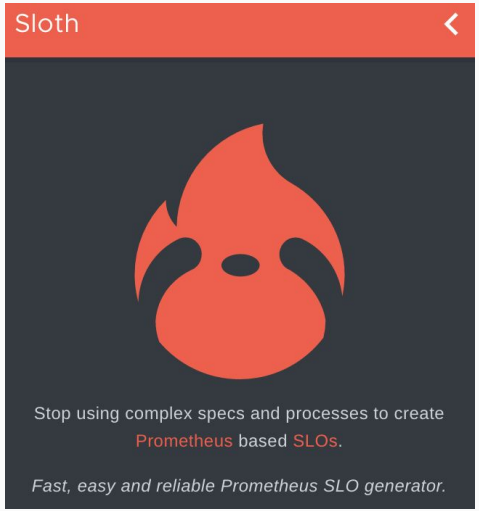
@dnsmichi

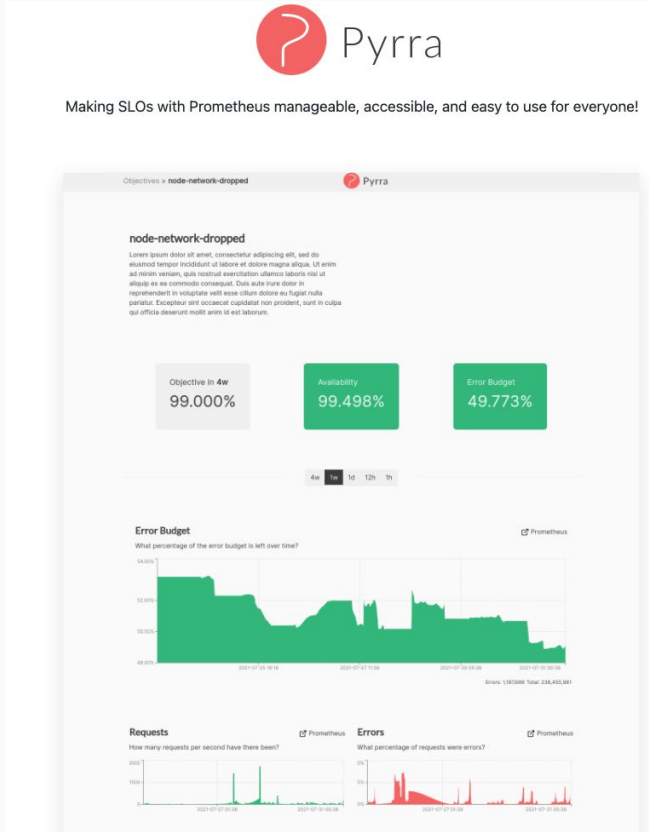
SLO spec - OpenSLO

SLO generators - Sloth

SLO management - Pyrra







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

Ops Iteration

Many dashboards & alerts

Learning & Documentation

Customizations

Onboarding

Goals

Immediately see what's important during an incident

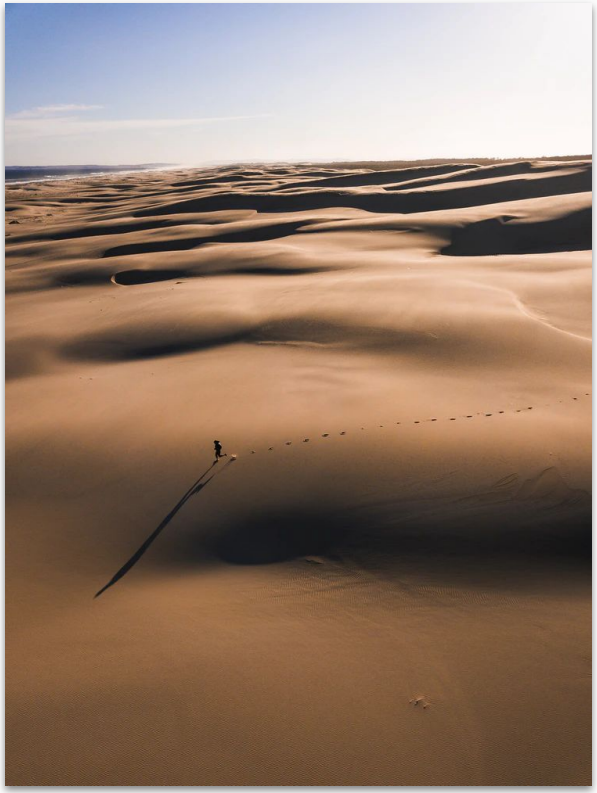


Photo by Silas Baisch on Unsplash

Customize kube-prometheus

Use jsonnet

Develop rules and dashboards

Monitor other namespaces

Add applications

customize.jsonnet — my-kube-prometheus

customize.jsonnet

```jsonnet
local kp = (import 'kube-prometheus/main.libsonnet') + {
  _config+:: {
    namespace: 'monitoring',

    prometheus+:: {
      namespaces: ["default", "kube-system", "ecc"],
    },
  },
};

{ ['00namespace-' + name]: kp.kubePrometheus[name] for name in std.objectFields(kp.kubePrometheus) } +
{ ['0prometheus-operator-' + name]: kp.prometheusOperator[name] for name in std.objectFields(kp.prometheusOperator) } +
{ ['node-exporter-' + name]: kp.nodeExporter[name] for name in std.objectFields(kp.nodeExporter) } +
{ ['kube-state-metrics-' + name]: kp.kubeStateMetrics[name] for name in std.objectFields(kp.kubeStateMetrics) } +
{ ['alertmanager-' + name]: kp.alertmanager[name] for name in std.objectFields(kp.alertmanager) } +
{ ['prometheus-' + name]: kp.prometheus[name] for name in std.objectFields(kp.prometheus) } +
{ ['grafana-' + name]: kp.grafana[name] for name in std.objectFields(kp.grafana) }

```

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

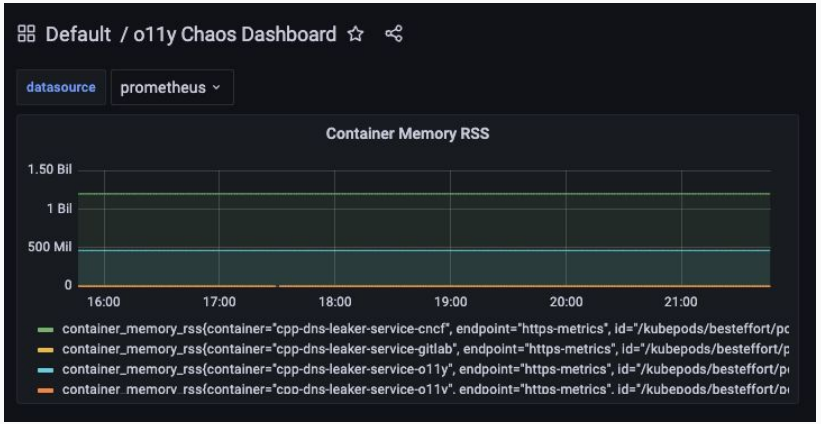
Custom dashboards

@dnsmichi

o11y-chaos-dashboard.jsonnet 857 bytes

```jsonnet
local grafana = import 'grafonnet/grafana.libsonnet';
local dashboard = grafana.dashboard;
local row = grafana.row;
local prometheus = grafana.prometheus;
local template = grafana.template;
local graphPanel = grafana.graphPanel;

{
  'o11y-chaos-dashboard.json':
    dashboard.new('o11y Chaos Dashboard')
    .addTemplate(
      {
        current: {
          text: 'Prometheus',
          value: 'Prometheus',
        },
        hide: 0,
        label: null,
        name: 'datasource',
        options: [],
        query: 'prometheus',
        refresh: 1,
        regex: '',
        type: 'datasource',
      },
    )
    .addRow(
      row.new()
      .addPanel(graphPanel.new('Container Memory RSS', span=6, datasource='$datasource')
                .addTarget(prometheus.target('container_memory_rss{container=~"cpp-dns-leaker-service.*"}')))
    ),
}
```

```jsonnet
      grafana+: {
        dashboards+:: (import 'o11y-chaos-dashboard.jsonnet'),
      },
```



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

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Wait, more than metrics?

Metrics

Logs/Events

Traces

Profiling

Error tracking

<more event types>

… is it Observability?

Photo by Eric Masur on Unsplash

https://about.gitlab.com/blog/2022/06/14/observability-vs-monitoring-in-devops/

Logs - too many decisions

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/

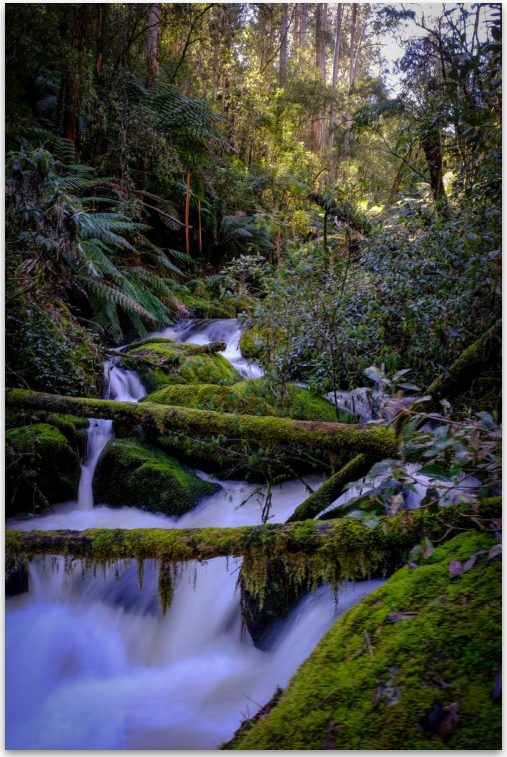
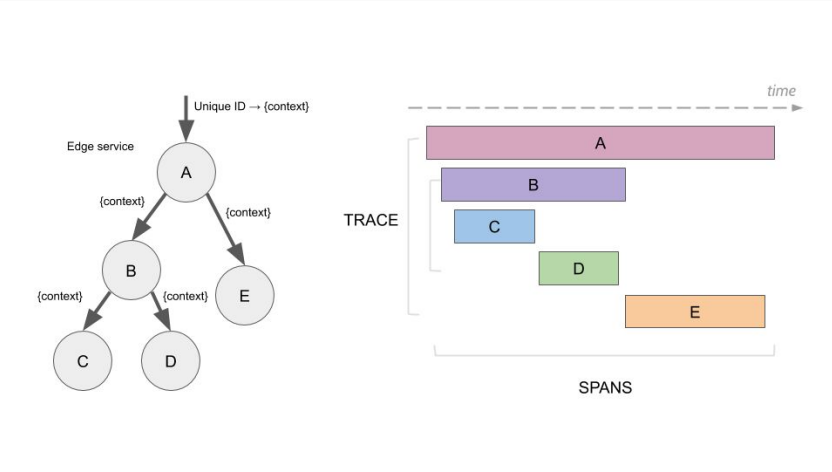


Photo by Zac Porter on Unsplash

Tracing

- 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/1MAVFeSsTNVWC9wPGOlg83wh8GFtR9hPbdVHuumtgOWA/edit

Image credit: Jaeger Tracing Docs

OpenTelemetry

Specification and framework beyond vendors

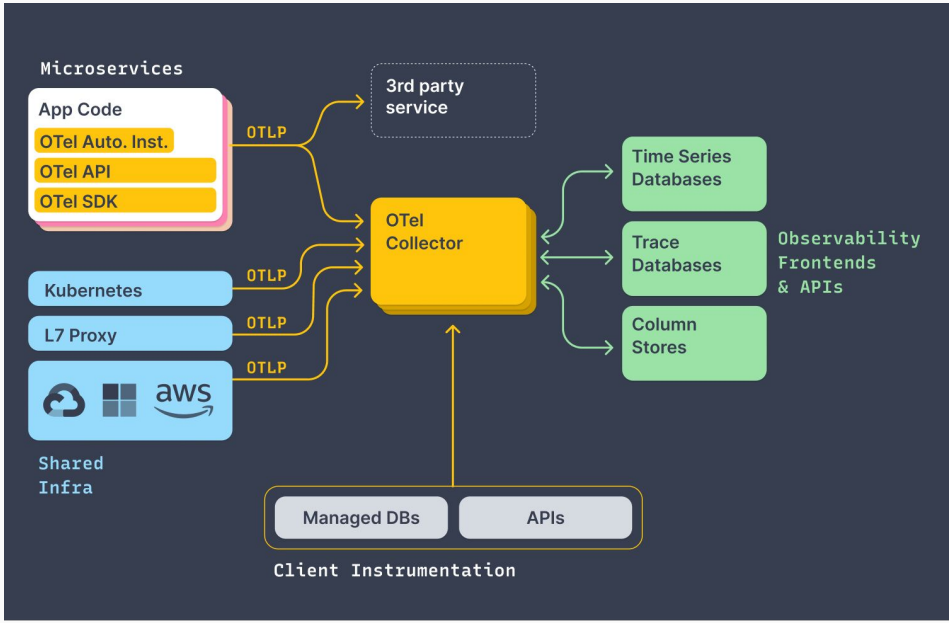
Collector + Sidecar

Bring your own backend

- Jaeger for Traces
- Prometheus for Metrics

Client libraries & SDKs

- Manual instrumentation (C++, Go, etc.)
- Auto-instrumentation (Java)



https://opentelemetry.io/docs/instrumentation/go/getting-started/

Image credit: OpenTelemetry Docs

Focus: Traces

- Kubernetes
  - Components send traces
  - App instrumentation with traces
    - Send to OpenTelemetry collector
      - Store in Jaeger, etc.
        - Visualize, correlate, alert

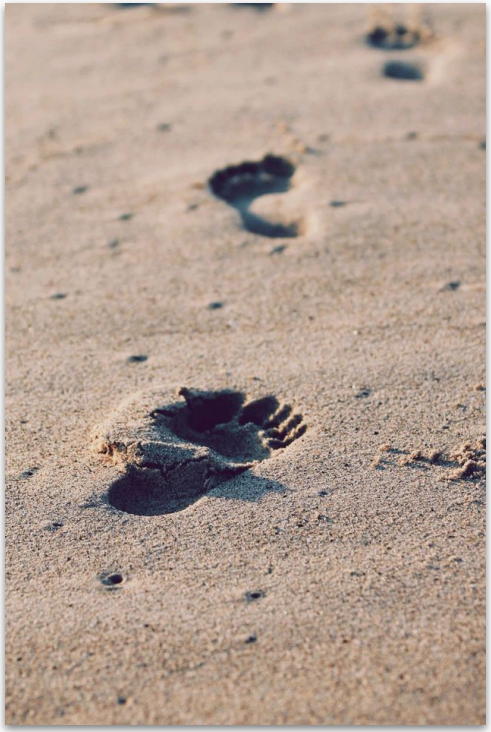


Photo by Christopher Sardegna on Unsplash

Traces help analyse slow requests

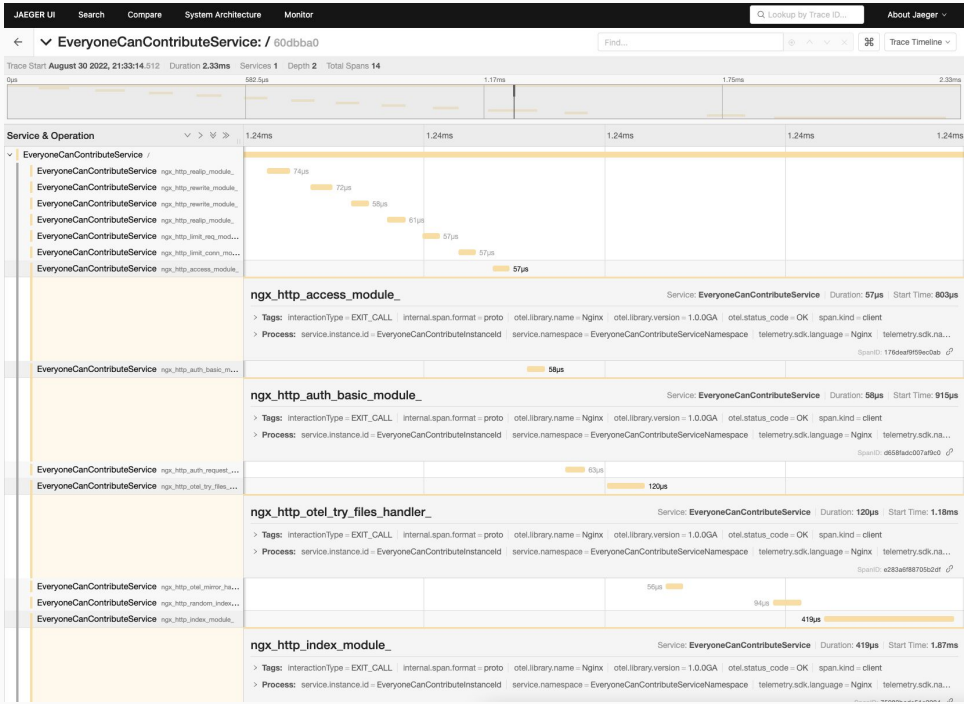
@dnsmichi

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

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

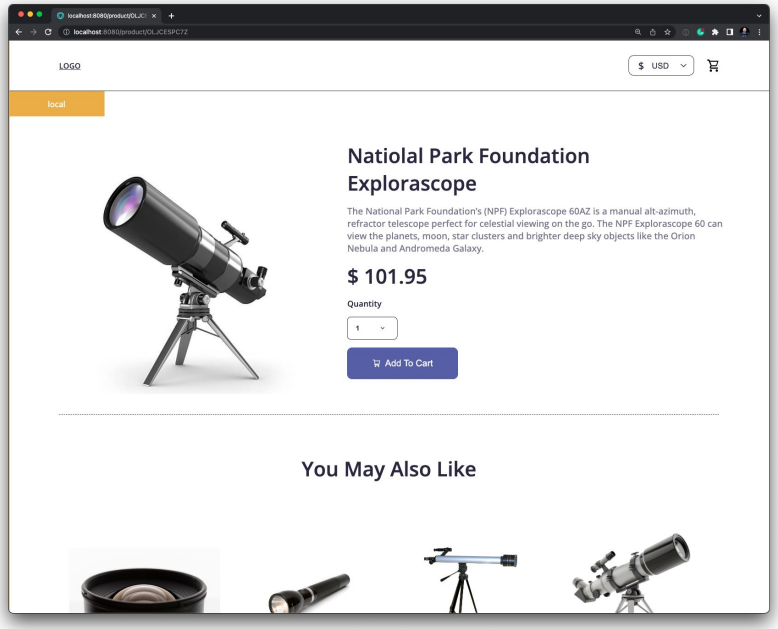
```
helm repo add open-telemetry
https://open-telemetry.github.io/opentelemetry-helm-charts

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 &
```

https://github.com/open-telemetry/opentelemetry-demo/blob/main/docs/kubernetes_deployment.md

EXERCISE: Inspect the shop website

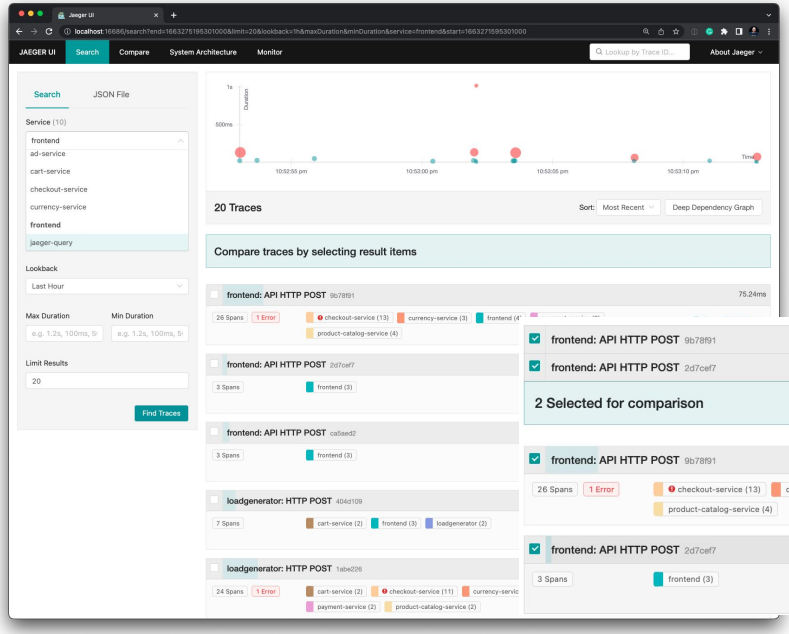
1. Navigate into the shop website in http://localhost:8080
    a. Try different paths and checkouts

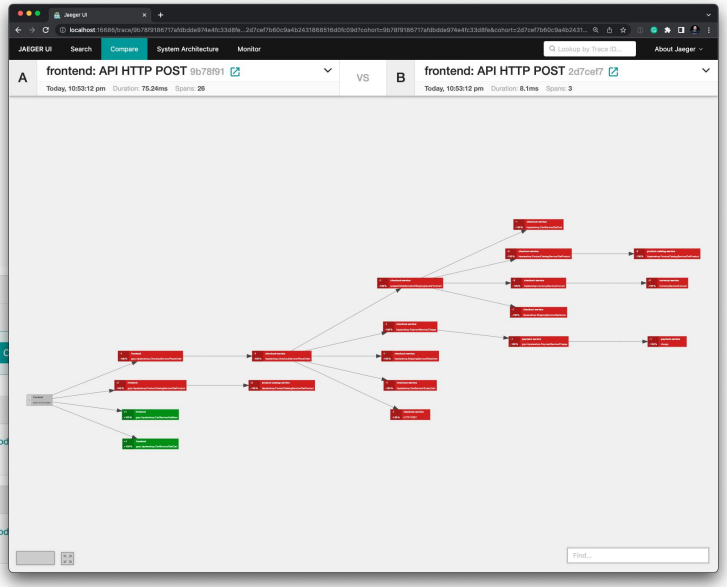


EXERCISE: Analyse traces in Jaeger

@dnsmichi

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





Discussion: Why traces?

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.

What's next?

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

https://opentelemetry.io/docs/instrumentation/

EXERCISE: Logs?

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

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

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

Photo by [Michael Friedrich]

Kubesimplify | Workshops

@dnsmichi

Data retention

Metrics

- Retention, Compaction

Traces

- Sampling

Logs

- Archiving

“What timeframe do you need for debugging incidents?”

Compliance requirements: “Archive everything on a magnet tape”

https://prometheus.io/docs/prometheus/latest/storage/

Storage: Distributed and long-term

Thanos for High Availability and long term storage

Cortex for distributed clusters, aggregated queries over Prometheus nodes

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/

Discussion: Resources and scaling

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?

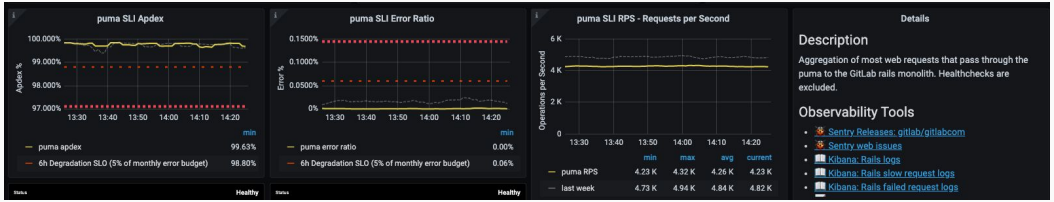
Manage everything yourself, or send data to SaaS platforms?

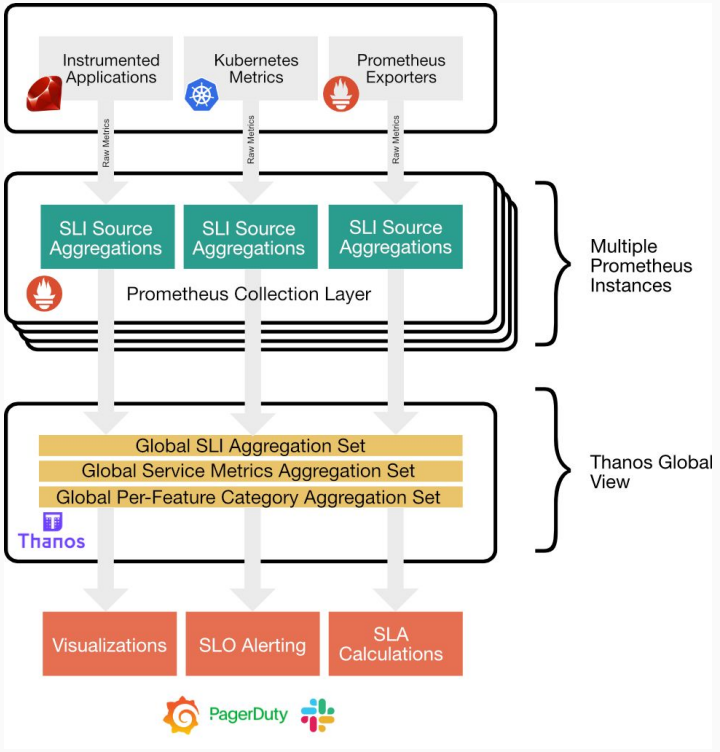
Discussion: Distributed scaling

- 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)

GitLab.com SaaS insights

- 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/gl-infra/gitlab-monitoring/-/tree/main/gitlab-monitoring

Observability and Chaos Engineering

Automate breaking things

Benefit from confidence with alerting and Observability

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Simulate incidents?

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

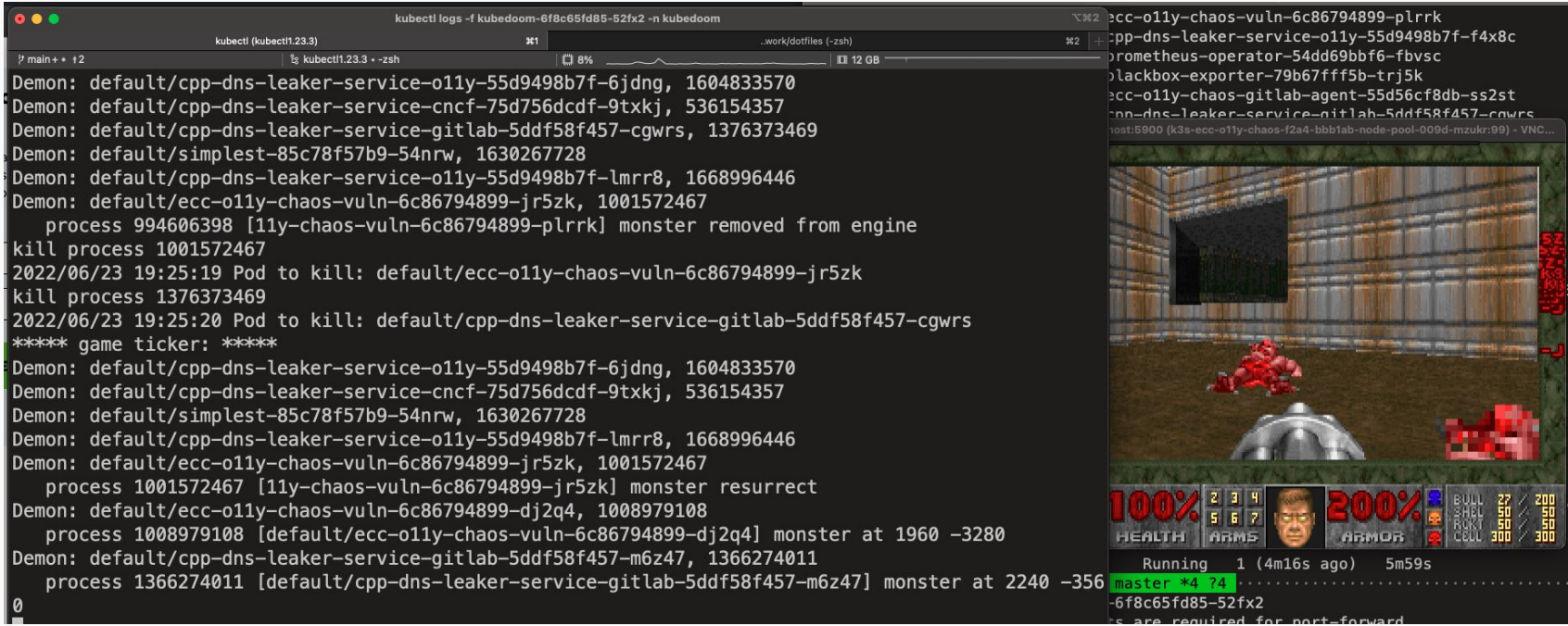
Instrumentation SDKs

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

Break more in a fun way

@dnsmichi

Kubedoom (might not be applicable for $work)



https://gitlab.com/everyonecancontribute/observability/k8s-o11y-chaos#game-kubedoom

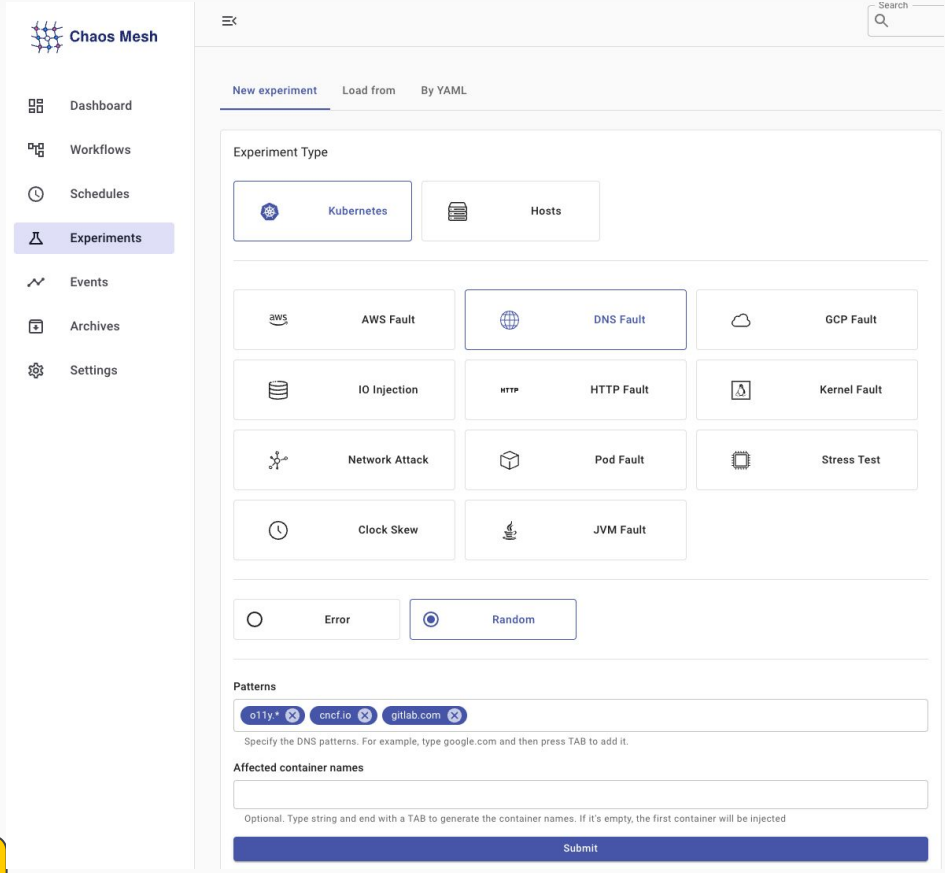
Chaos Mesh

Fail Kubernetes or Hosts

- Pods, network, http, time, DNS, …

Run Chaos Experiments

- Once
- Continuously as schedule

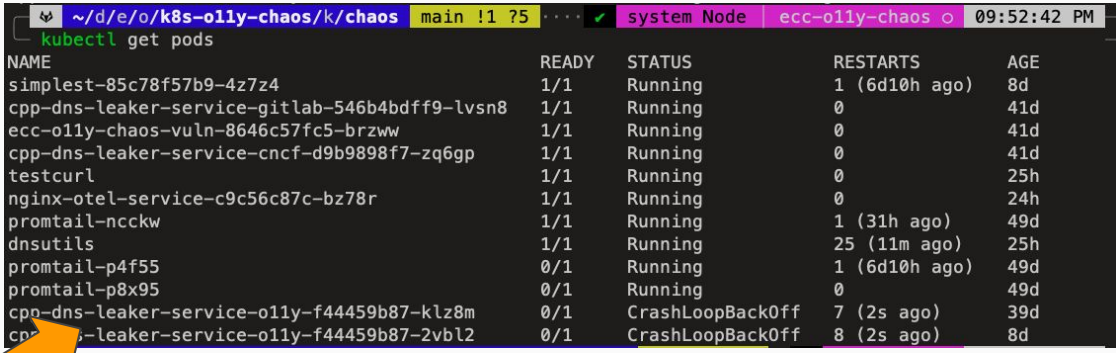


https://chaos-mesh.org/docs/quick-start/
https://chaos-mesh.org/docs/simulate-dns-chaos-on-kubernetes/

Generate some chaos

pod-fail-schedule.yaml 440 bytes

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: pod-kill-cpp-dns-leaker
spec:
  schedule: '* * * * * '
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: PodChaos
  podChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: cpp-dns-leaker-service-o11y
    mode: all
    action: pod-failure
    duration: 10s
    gracePeriod: 0
```



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://songrgg.github.io/operation/how-to-alert-for-Pod-Restart-OOMKilled-in-Kubernetes/

It’s always DNS

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        }
```

https://gitlab.com/everyonecancontribute/observability/cpp-dns-leaker
More details in my Kubecon EU talk: https://www.youtube.com/watch?v=BkREMg8adaI

Image generated with https://gitlab.com/gitlab-de/go-excusegen which itself is based on the XKCD comic at https://xkcd.com/303/

EXERCISE (advanced): Chaos Mesh Setup 1,2,3

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.

EXERCISE (advanced): Deploy DNS resolver apps and Prometheus alerts

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
```

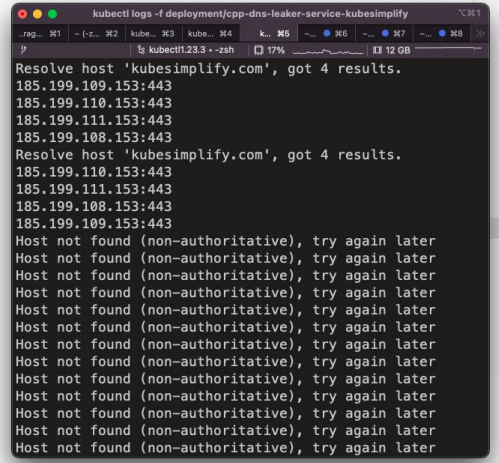
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/cpp-dns-leaker/deployment.yaml
https://gitlab.com/everyonecancontribute/workshops/kube-simplify/k8s-o11y-2022/-/blob/main/kubernetes/o11y/cpp-dns-leaker/prometheus-rule-memory.yml

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
```



New Schedule | Load from | By YAML

Create an experiment or schedule by filling in or uploading YAML

```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:
      - o11y.*
      - cncf.io
      - gitlab.com
      - containerdays.io
      - kubesimplify.com
  selector:
    namespaces:
      - default
  duration: '60s'
```

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

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

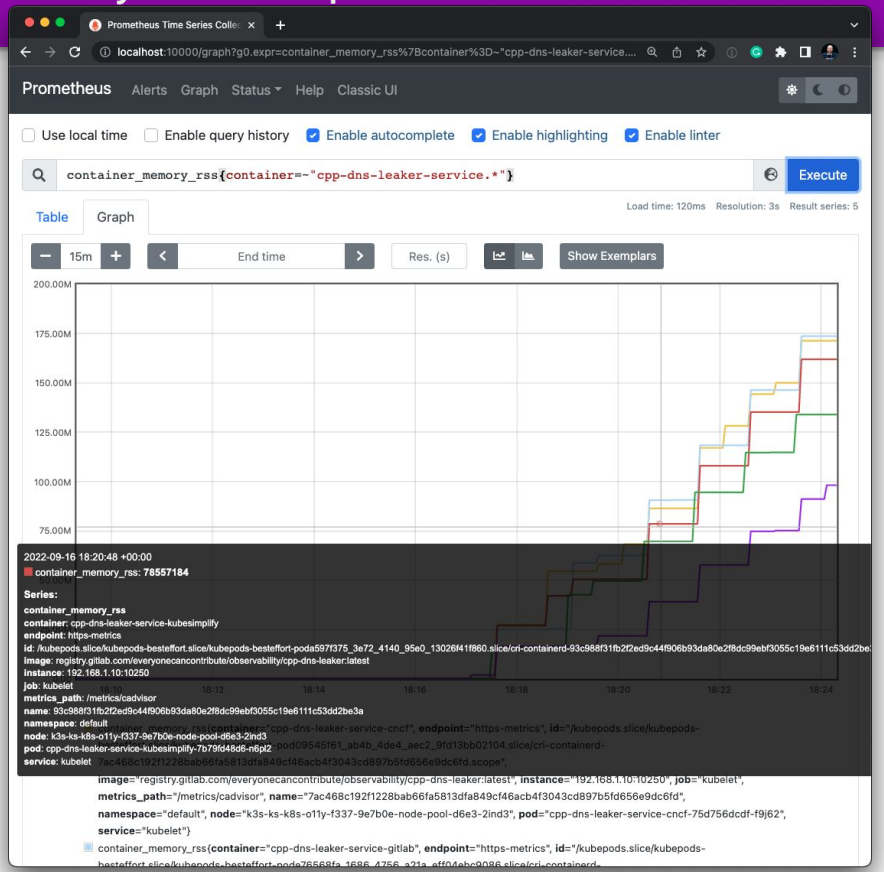
Demo hosted by Civo

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.*"}
```

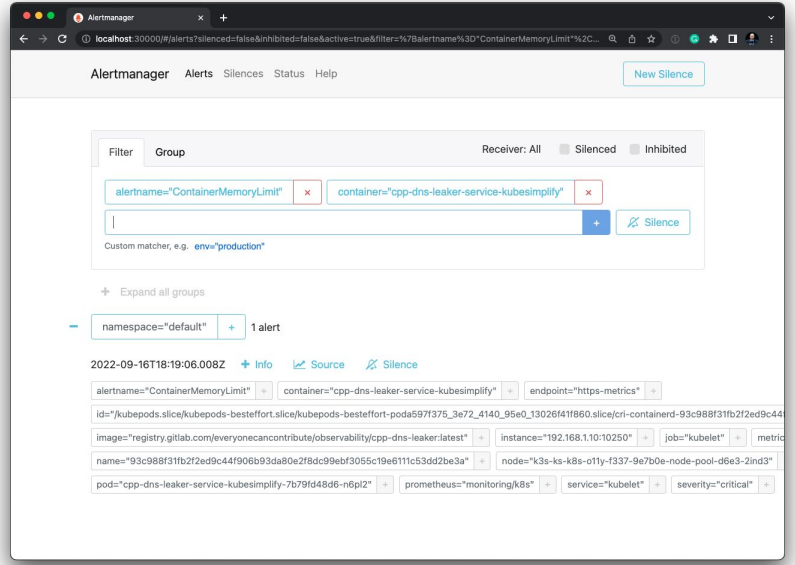


https://gitlab.com/everyonecancontribute/observability/cpp-dns-leaker
More details in my Kubecon EU talk: https://www.youtube.com/watch?v=BkREMg8adaI

EXERCISE (advanced): Verify Alerts

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
```



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

Chaos generated alerts and red dashboards

- 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

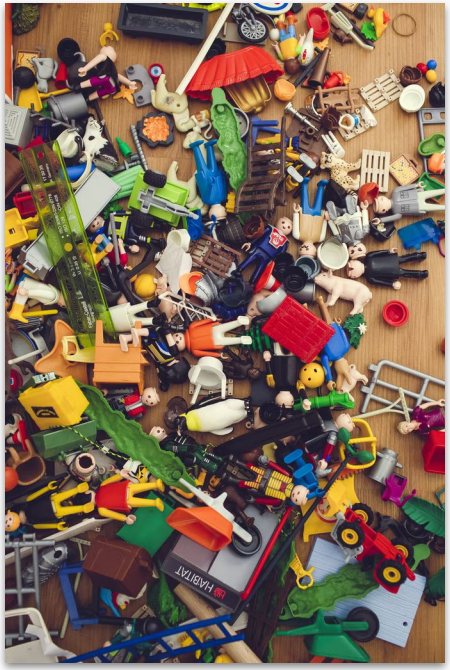


Photo by Markus Spiske on Unsplash

Traces show slow requests: Chaos

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.yml 514 bytes

```yaml
kind: Schedule
apiVersion: chaos-mesh.org/v1alpha1
metadata:
  namespace: default
  name: stress-cpu-mem-nginx-otel
spec:
  schedule: '* * * * * '
  startingDeadlineSeconds: null
  concurrencyPolicy: Forbid
  historyLimit: 1
  type: StressChaos
  stressChaos:
    selector:
      namespaces:
        - default
      labelSelectors:
        app: nginx-otel-service
    mode: all
    stressors:
      memory:
        workers: 10
        size: '100'
      cpu:
        workers: 10
        load: 100
    duration: 60s
```

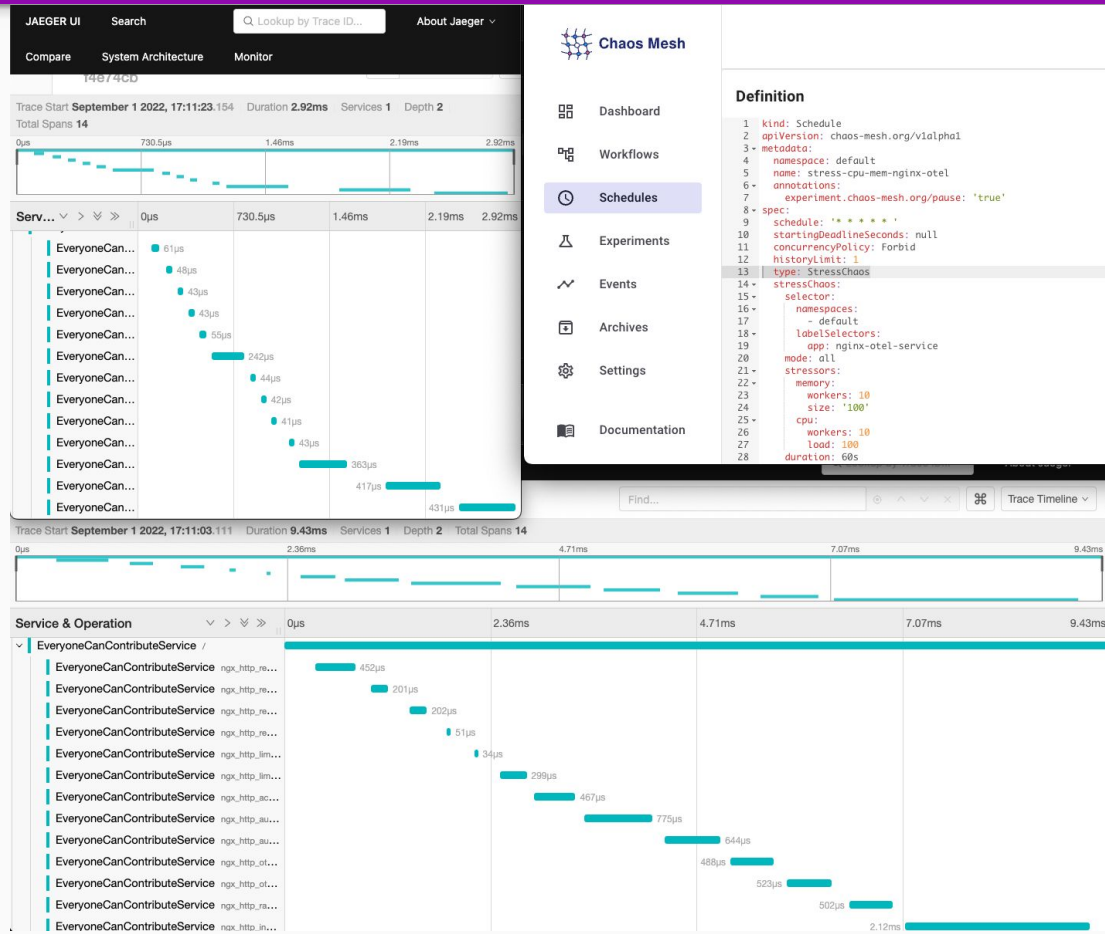
Traces show slow requests: Chaos

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

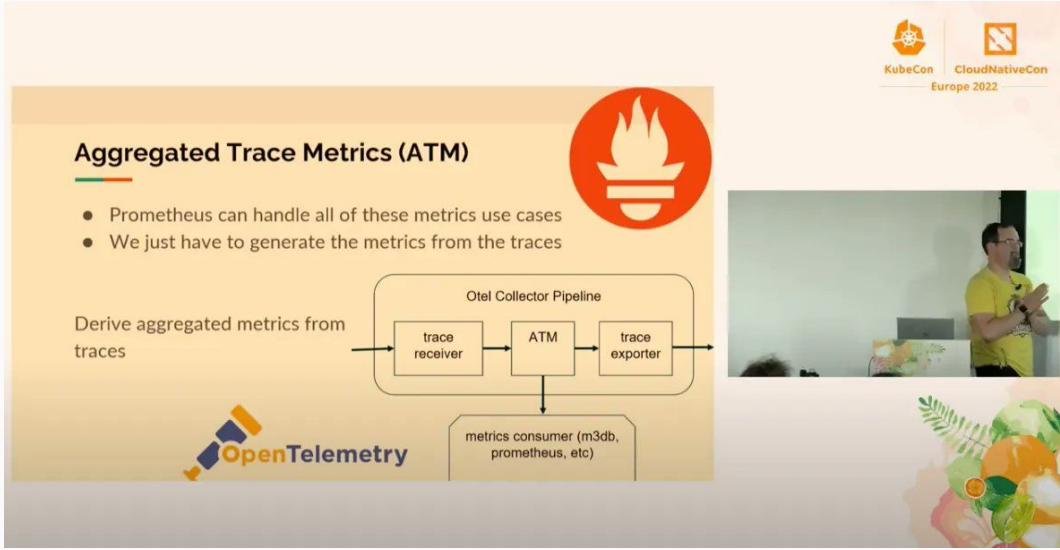
Ideas

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

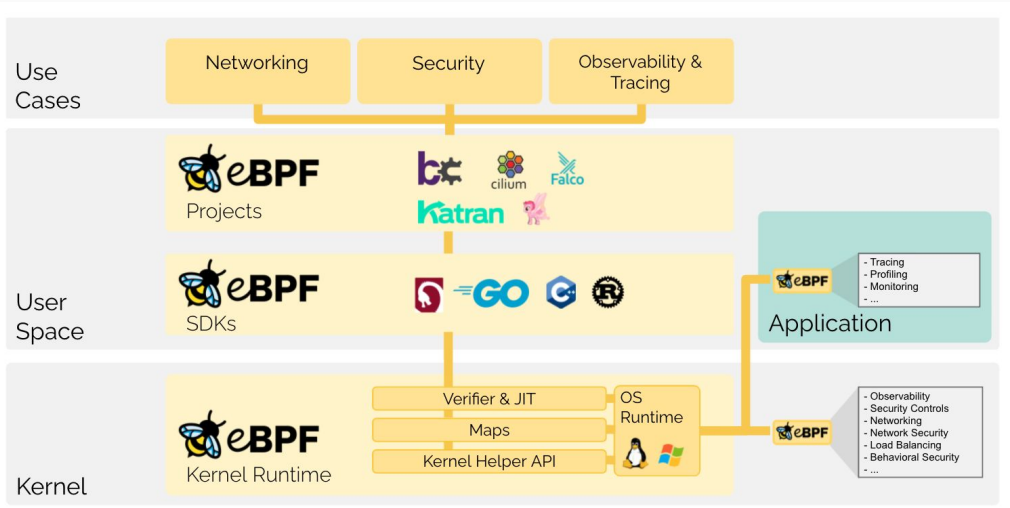
Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

“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

Where does eBPF fit into Kubernetes Observability?

Does it complement / integrate with Prometheus metrics?

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

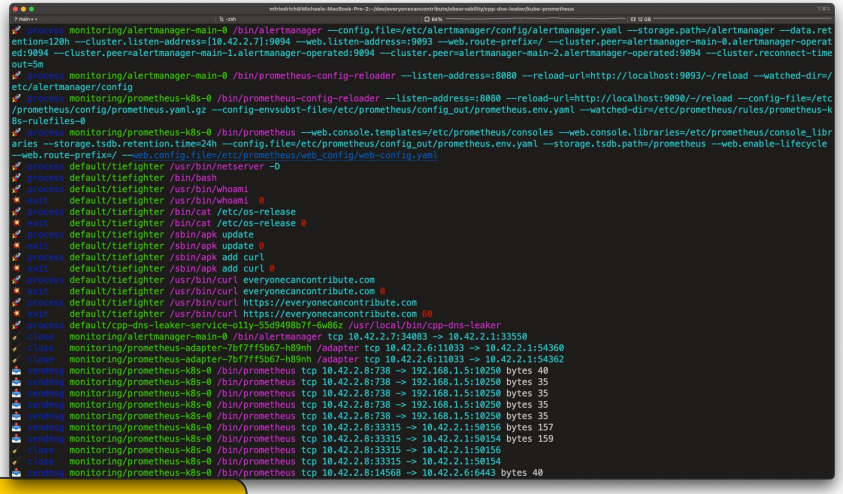
https://medium.com/@isalapiyarisi/getting-started-on-kubernetes-observability-with-ebpf-88139eb13fb2 (image credit)

Observability with eBPF

Cilium Tetragon

Pixie

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



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

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

SLOs with Prometheus + Quality Gates with Keptn

Metrics & Alerts calculation

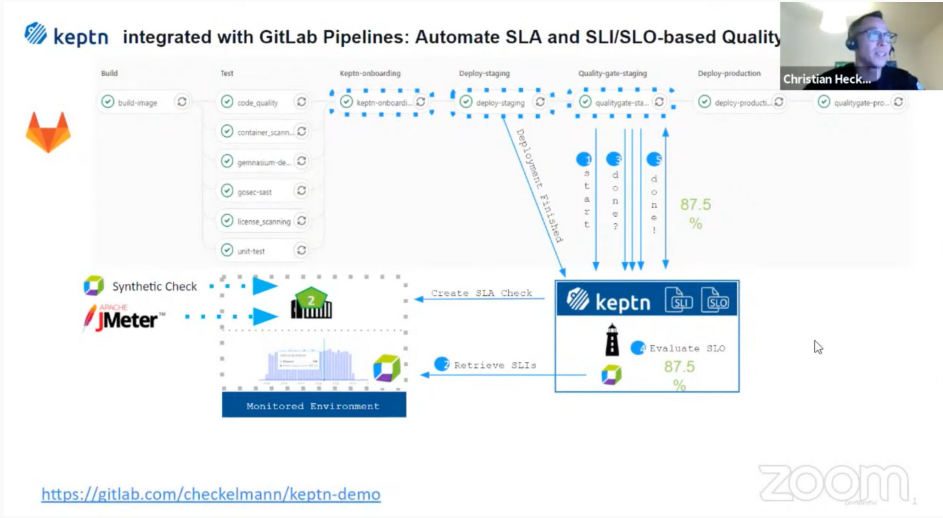
PromQL queries in OpenSLO format

CI/CD

Environments

Metrics Monitoring

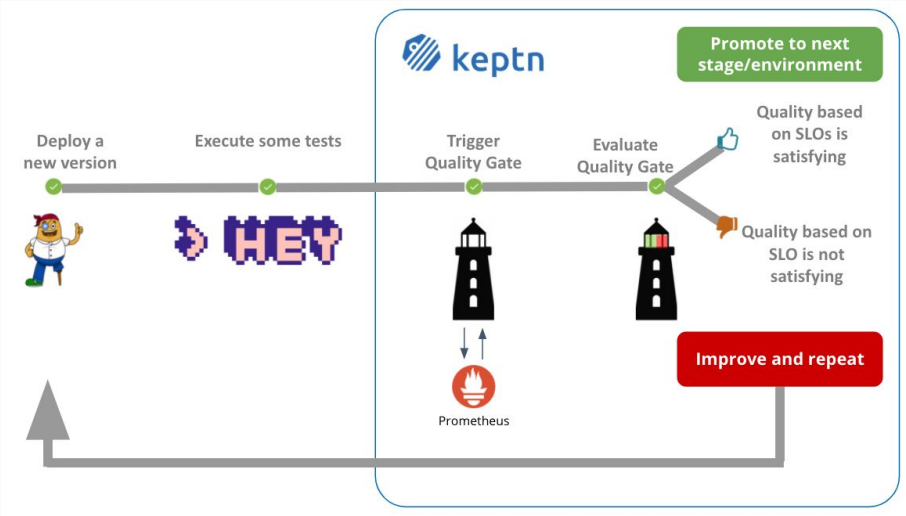
Keptn with SLOs for quality gates

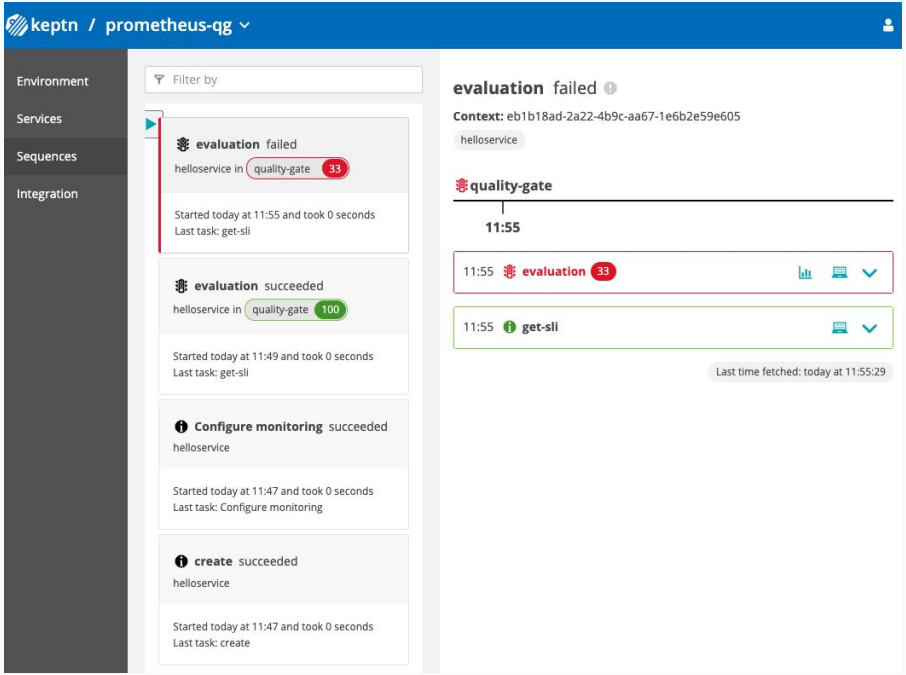


https://everyonecancontribute.com/post/2020-11-11-cafe-8-keptn/

Screenshot by Michael Friedrich

Oh Keptn, my Keptn





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

Security

Kubesimplify | Workshops

@dnsmichi

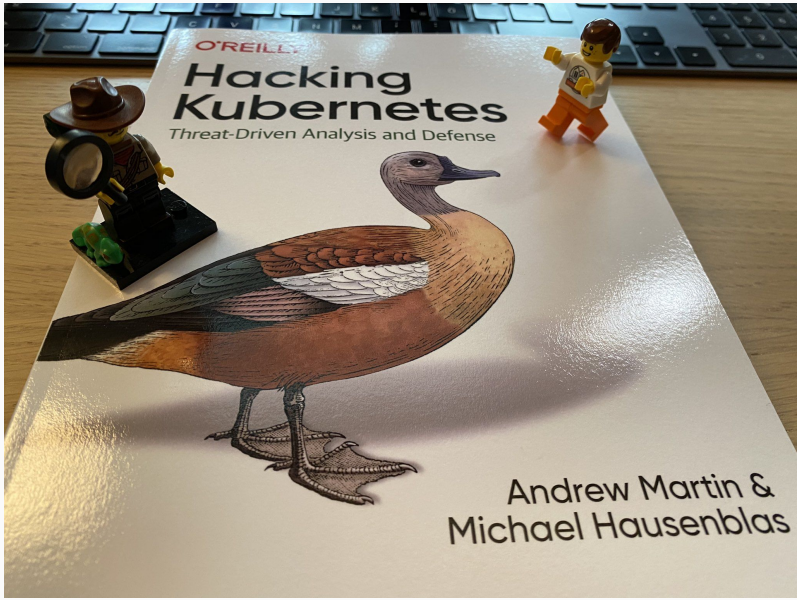
Kubernetes Security Monitoring

- 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/

Prometheus Hardening

- 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/

Hacking Kubernetes

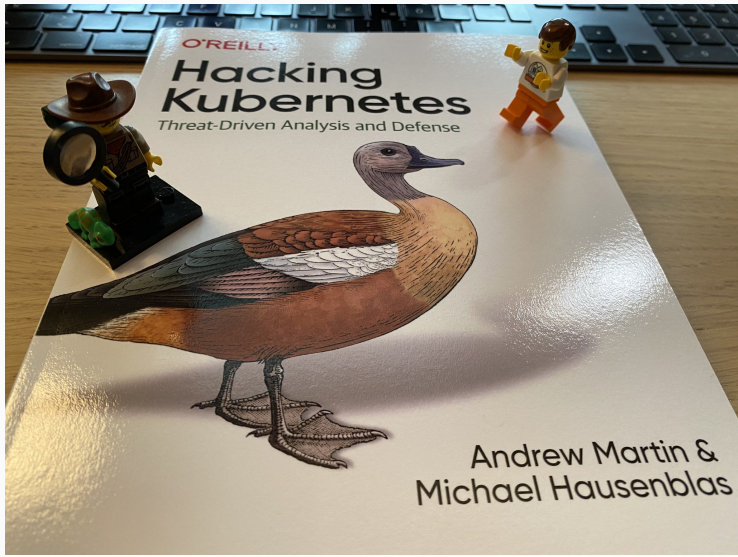
Security Defaults in Kubernetes need improvements

Impersonate the attacker

Use Chaos Engineering principles for pen testing?

Create a Chaos Experiment that downloads and installs malicious software

Observability for better Security?



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

Photo by Michael Friedrich

Your adventure

Photo by Michael Friedrich

Kubesimplify | Workshops

@dnsmichi

Learn app instrumentation? (this needs a workshop on its own ;-))

- 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/

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