You can see a walkthrough of how Kubernetes could run a flask sklearn app locally in Docker desktop here[199].

Kubernetes Summary

There are many compelling reasons to use Kubernetes. Let's summarize them:

- Created, Used, and Open Sourced by Google
- High availability architecture
- Auto-scaling is built-in
- Rich Ecosystem
- Service discovery
- Container health management
- Secrets and configuration management

Another advantage is that Kubernetes is cloud-agnostic, and it could be a solution for companies that are willing to take on the additional complexity to protect against "vendor lockin."

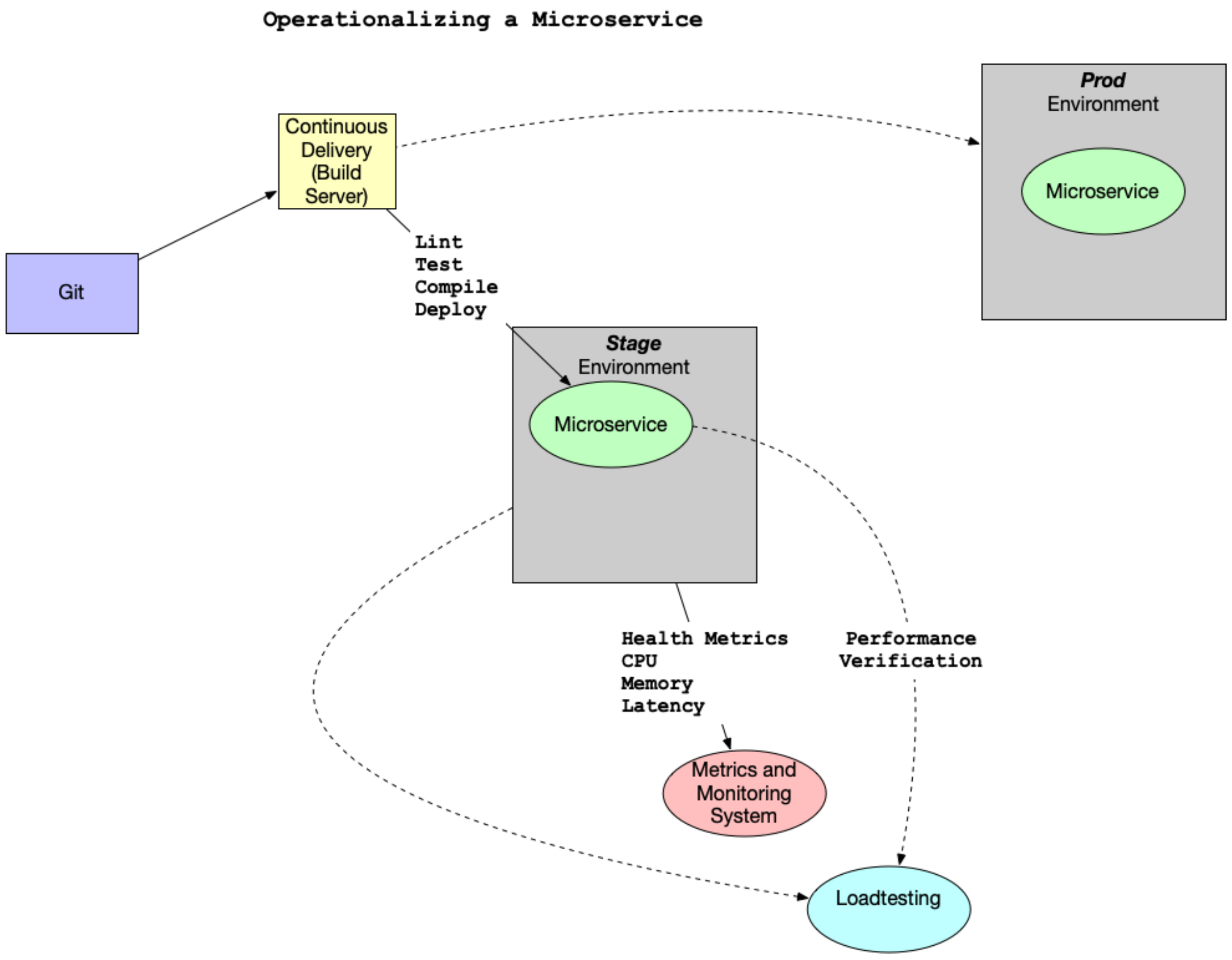
Begin reading here

Operationalizing a Microservice Overview

A critical factor in developing a microservice is to think about the feedback loop. In this diagram, a GitOps[200] style workflow implements.

- Application stored in Git
- Changes in Git trigger the continuous delivery server, which tests and deploys it to a new environment. This environment configures as code Infrastructure as Code (IaC).
- The microservice, which could be a containerized service. It runs in Kubernetes or a FaaS (Function as a Service) running on AWS Lambda. This microservice has logging, metrics, and instrumentation included.
- A load test using a tool like locust[201]

[199]https://github.com/noahgift/container-revolution-devops-microservices
[200]https://queue.acm.org/detail.cfm?id=3237207
[201]https://locust.io/



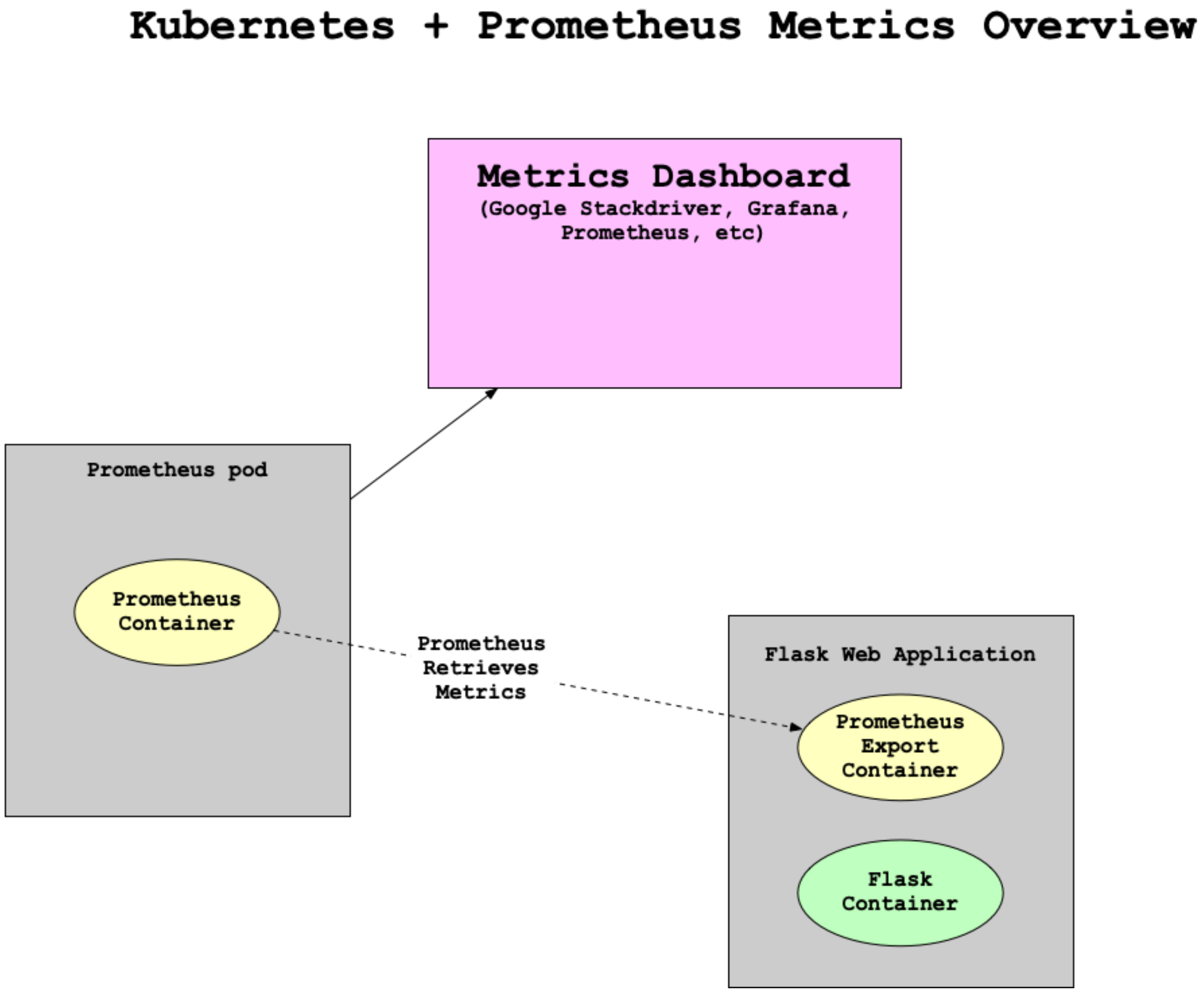
operationalize-microservice

- When the performance and auto-scaling is verified, the code is merged to production and deployed

What are some of the items that could be alerted on with Kubernetes?

- Alerting on application layer metrics
- Alerting on services running on Kubernetes
- Alerting on the Kubernetes infrastructure
- Alerting on the host/node layer

How could you collect metrics with Kubernetes and Prometheus? Here is a diagram that walks through a potential workflow. A few things to note here are that are two pods. One pod attaches to the Prometheus collector, and the second pod has a "sidecar" Prometheus container that sits alongside the Flask application. This process will propagate up to a centralized monitoring system that visualizes the clusters' health and trigger alerts.


Kubernetes + Prometheus Metrics Overview
Metrics Dashboard
(Google Stackdriver, Grafana, Prometheus, etc)
Prometheus pod
Prometheus Container
Prometheus Retrieves Metrics
Flask Web Application
Prometheus Export Container
Flask Container

prometheus-kubernetes

Another helpful resource is an official sample project from Google Cloud Monitoring apps running on multiple GKE clusters using Prometheus and Stackdriver[202]

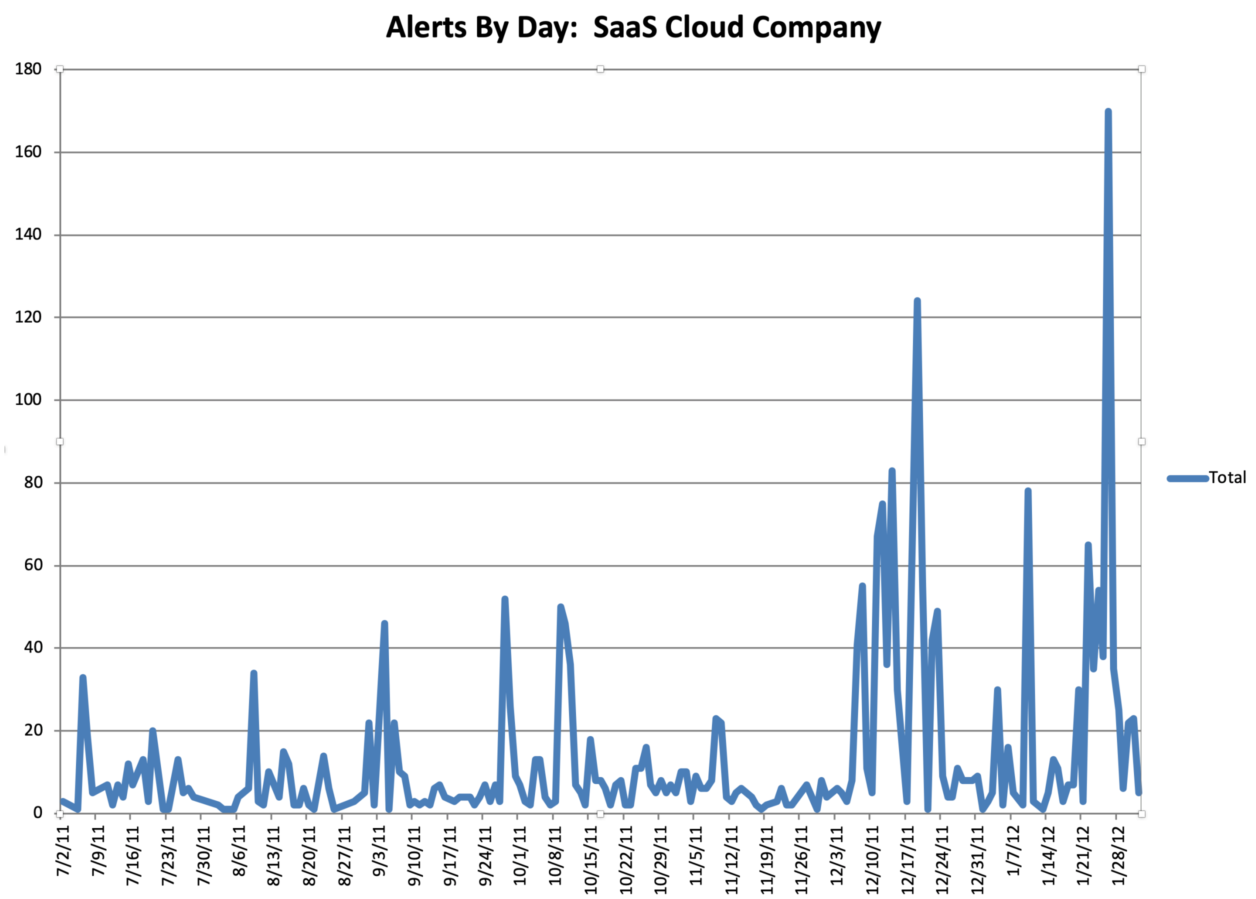
Creating effective alerts

At one company I worked at, a homegrown monitoring system (again initially created by the founders) alerted on average every 3-4 hours, 24 hours a day.

Because everyone in engineering except the CTO was on call, most of the engineering staff was always sleep deprived because it guaranteed that there were alerts about the system not working every night. The "fix" to the omens was to restart services. I volunteered to be on call for one month straight to allow engineering the time to fix the problem. This sustained period of suffering and lack

[202]https://cloud.google.com/solutions/monitoring-apps-running-on-multiple-gke-clusters-using-prometheus-and-stackdriver

of sleep led me to realize several things. One, the monitoring system was no better than random. I could potentially replace the real Voodoo with a random coin flip.



prometheus-kubernetes

Even more distressing in looking at the data was that engineers had spent YEARS of their life responding to pages and getting woken up at night, and it was utterly useless. The suffering and sacrifice accomplished nothing and reinforced the sad truth that life is not fair. The unfairness of the situation was quite depressing, and it took quite a bit of convincing to get people to agree to turn off the alerts. There is a built-in bias in human behavior to continue to do what you have always done. Additionally, because the suffering was so severe and sustained, there was a tendency to attribute a deeper meaning. Ultimately, it was a false God. *Reference: Gift, Noah (2020) Python for DevOps: pg. 226*

Learn how to use Alerts and Monitoring in the following screencast.

Video Link: https://www.youtube.com/watch?v=xrXjgtOSX6o[203]

[203]https://www.youtube.com/watch?v=xrXjgtOSX6o

Five-Why's and Kaizen

One way our troubled company could have swapped Voodoo for a sane alert process was to use the Five Why's method. In a nutshell, it originated from Kaizen, a process of continuous improvement, from the Japanese Automobile industry post-WWII. The Five Why's strategy is to keep asking questions until the root cause appears.

Learn about the Five Whys in the following screencast.

Video Link: https://www.youtube.com/watch?v=9jS3cwjIJEo[204]

Learn about Continuous Improvement in the following screencast.

Video Link: https://www.youtube.com/watch?v=mZVbUbYwFQo[205]

Alerting Statistical Theory

A good way to think about alerts is to think about a normal distribution from Statistics. Consider a normal distribution[206], "six sigma[207]" and the 68-95-99.7 rule[208]. Computer systems events are often normally distributed, meaning that all events within three standard deviations from the mean occur with 99.7 occurrences.

Design a process that alerts senior engineers when events are more significant than three standard deviations from the mean and write up how the alerts should work, i.e.

- Who should get a page when an event is more significant than three standard deviants from the mean?
- Should there be a backup person who gets alerted if the first person doesn't respond within five minutes?

*Should an alert wake up a team member at one standard deviation? What about two?

Learn to build an alert from scratch in the following screencast.

Video Link: https://www.youtube.com/watch?v=8cl_ZbmgDhw[209]

Getting Started with Prometheus

Prometheus[210] is a commonly used metrics and alerting solution typically with containers and Kubernetes. To run this tutorial, do the following.

[204]https://www.youtube.com/watch?v=9jS3cwjIJEo
[205]https://www.youtube.com/watch?v=mZVbUbYwFQo
[206]https://en.wikipedia.org/wiki/Normal_distribution
[207]https://en.wikipedia.org/wiki/Six_Sigma_
[208]https://en.wikipedia.org/wiki/68%E2%80%9395%E2%80%9399.7_rule
[209]https://www.youtube.com/watch?v=8cl_ZbmgDhw
[210]https://prometheus.io/

- Use a local environment or preferably AWS Cloud9. If you use AWS Cloud9, you will need to expose port 9090 via EC2 Security Group.
- Download, install[211] and run Prometheus. On AWS Cloud9, you would install the latest release with `*.linux-amd64.tar.gz` in the name. This would like something like `wget <some release>.linux-amd64.tar.gz`.

```
tar xvfz prometheus-*.tar.gz
cd prometheus-*
```

- Configure Prometheus by creating a `prometheus.yml` file

```
global:
  scrape_interval:     15s
  evaluation_interval: 15s

rule_files:
  # - "first.rules"
  # - "second.rules"

scrape_configs:
  - job_name: prometheus
    static_configs:
      - targets: ['localhost:9090']
```

- Start Prometheus

Wait about 30 seconds for Prometheus to collect data.

```
./prometheus --config.file=prometheus.yml
```

- View data through the expression browser

Go to http://localhost:9090/graph[212]. Choose the "console" within the graph. One metric that Prometheus collects is how many times http://localhost:9090/metrics[213] invokes. If you refresh a few times, then type the following in the expression console, you can see a time series result.

[211]https://prometheus.io/download/
[212]http://localhost:9090/graph
[213]http://localhost:9090/metrics

~~~
* View data through the graphing interface

Another way to view data is via the graphing interface.  Go to [http://localhost:909\
0/graph](http://localhost:9090/graph).  Use the "Graph" tab.

```rate(promhttp_metric_handler_requests_total{code="200"}[1m])```

* *(OPTIONAL)* Going further, feel free to experiment with how that would work by fo\
llowing the example below.

A more sophisticated example would [involve also collecting](https://prometheus.io/d\
ocs/prometheus/latest/getting_started/#downloading-and-running-prometheus) data from\
 clients.  Next, download these `go` clients using the code below and run them.

```bash
# Fetch the client library code and compile the example.
git clone https://github.com/prometheus/client_golang.git
cd client_golang/examples/random
go get -d
go build

# Start 3 example targets in separate terminals:
./random -listen-address=:8080
./random -listen-address=:8081
./random -listen-address=:8082
~~~

Next, add these clients in the `prometheus.yml`

```
scrape_configs:
  - job_name:       'example-random'

    # Override the global default and scrape targets from this job every 5 seconds.
    scrape_interval: 5s

    static_configs:
      - targets: ['localhost:8080', 'localhost:8081']
        labels:
          group: 'production'

      - targets: ['localhost:8082']
        labels:
          group: 'canary'
```

Restart Prometheus and see these metrics in the expression browser.

`rpc_durations_seconds`.

Based on guide from official Prometheus documentation[214] *and guide here*[215]

Learn to use Prometheus in the following screencast.

Video Link: https://www.youtube.com/watch?v=4bcBS1G3GWI[216]

Creating a Locust Load test with Flask

One powerful way to create a simple load test is with Locust and Flask. Here is an example of a simple flask hello world app. Here is source code[217] for reference.

```python
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'

if __name__ == "__main__":
    app.run(host='0.0.0.0', port=8080, debug=True)
```

The load test file is straightforward to configure. Notice the `index` function calls into the `main`, and only flask route.

```python
from locust import HttpLocust, TaskSet, between

def index(l):
    l.client.get("/")

class UserBehavior(TaskSet):
    tasks = {index: 1}

class WebsiteUser(HttpLocust):
    task_set = UserBehavior
    wait_time = between(5.0, 9.0)
```

[214]https://github.com/prometheus/docs/blob/432af848c749a7efa1d731f22f73c27ff927f779/content/docs/introduction/first_steps.md
[215]https://prometheus.io/docs/prometheus/latest/getting_started/#downloading-and-running-prometheus
[216]https://www.youtube.com/watch?v=4bcBS1G3GWI
[217]https://github.com/noahgift/docker-flask-locust