Node.js in the Cloud

...

Red Hat Node.js Team

Welcome 👋

- Introduction
- Overview of cloud native concepts
- Workshop

Who we are?

- **Bethany Griggs -** GitHub: @BethGriggs
- **Luke Holmquist -** Twitter: @sienaluke, GitHub: @lholmquist
- **Michael Dawson -** Twitter: @mhdawson1, GitHub: @mhdawson

Special Mention - Many thanks for all the work he put into the tutorial!

- **Costas Papastathis -** Twitter: @pacostas1, GitHub: @pacostas

Red Hat Node.js Team

Cloud Native Workshop

- Introduction to cloud-native development with Node.js.
- Walking you through how to extend an Express.js-based application to leverage some key cloud capabilities.
- The workshop will cover key concepts and technologies, including:
  - Health checks
  - Metrics
  - Containers (Docker/Podman)
  - Kubernetes
  - Prometheus
  - Grafana

Tutorial Prerequisites

https://github.com/nodeshift/tutorial/tree/main/cloud-native#prerequisites

Tutorial - https://red.ht/nodeconf2022

What are cloud native applications?

Cloud Native Landscape

v20180425

See the interactive landscape at l.cncf.io

Greyed logos are not open source

App Definition and Development

Database and Data Warehouse

Vitess (CNCF Incubating)

Streaming

NATS (CNCF Incubating)

Source Code Management

Application Definition and Image Build

Continuous Integration / Continuous Delivery (CI/CD)

Orchestration & Management

Scheduling & Orchestration

kubernetes (CNCF Graduated)

Coordination & Service Discovery

CoreDNS (CNCF Incubating)

Service Management

gRPC (CNCF Incubating)

envoy (CNCF Incubating)

LINKERD (CNCF Incubating)

Open Policy Agent (CNCF Sandbox)

Runtime

Cloud-Native Storage

ROOK (CNCF Sandbox)

Container Runtime

containerd (CNCF Incubating)

rkt (CNCF Incubating)

Cloud-Native Network

CNI (CNCF Incubating)

Provisioning

Host Management / Tooling

Infrastructure Automation

Container Registries

Secure Images

Notary (CNCF Incubating)

TUF (CNCF Incubating)

Key Management

spiffe (CNCF Sandbox)

SPIRE (CNCF Sandbox)

Cloud

Public

Private

This landscape is intended as a map through the previously uncharted terrain of cloud native technologies. There are many routes to deploying a cloud native application, with CNCF Projects representing a particularly well-traveled path.

l.cncf.io

CLOUD NATIVE Landscape

CLOUD NATIVE COMPUTING FOUNDATION

Redpoint Amplify

Platforms

Certified Kubernetes - Distribution

Certified Kubernetes - Hosted

Certified Kubernetes - Installer

Non-Certified Kubernetes

PaaS/Container Service

Observability & Analysis

Monitoring

Prometheus (CNCF Incubating)

Logging

fluentd (CNCF Incubating)

Tracing

JAEGER (CNCF Incubating)

OPENTRACING (CNCF Incubating)

Serverless

Special

Kubernetes Certified Service Provider

Kubernetes Training Partner


CNCF LANDSCAPE EDITION
1000
PREMIUM PUZZLE
SOFTCLICK TECHNOLOGY
Ravensburger Puzzle
Overwhelmed? Please see the CNCF Trail Map. That and the interactive landscape are at l.cncf.io
CNCF Cloud Native Landscape
2019-04-21T02:20:46Z 7049999
l.cncf.io
overwhelmed? deal with it. Age 30-45 yo. CKA not included
NOW WITH INLETS AND K3S

Cloud Native Glossary

https://glossary.cncf.io/

Building Enterprise Cloud Native Applications

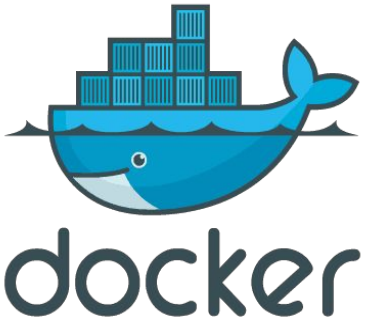
Building Enterprise Cloud Native Node.js Applications

- Module/dependency diligence
  - Consider licenses, security issues, maintenance, compatibility, support
- Functional Components
  - Web Framework, GraphQL, Caching, Authentication/Authorization
- Development
  - Code consistency, testing, proxying, etc.
- Operations
  - Health checks
  - Metrics

Reference Architecture for Node.js Applications

https://nodeshift.dev/nodejs-reference-architecture

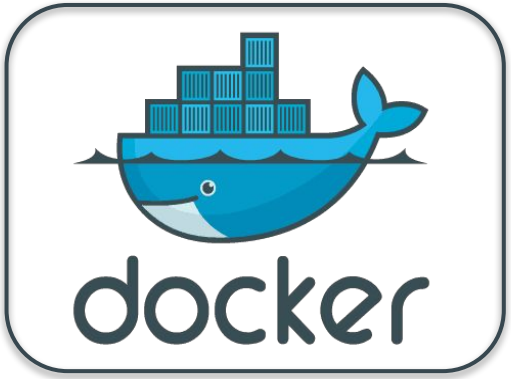
CLOUD NATIVE
COMPUTING FOUNDATION



kubernetes

Prometheus

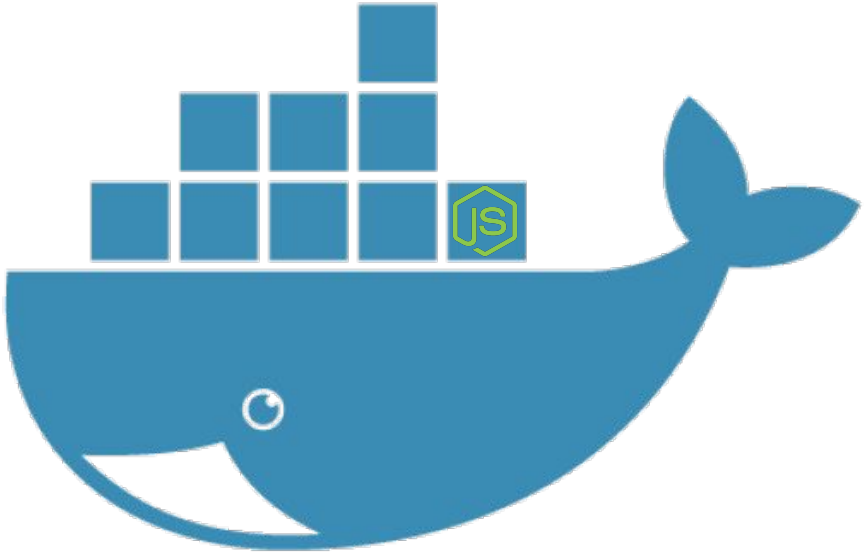
CLOUD NATIVE
COMPUTING FOUNDATION



kubernetes

Prometheus


docker



- Tool designed to make it easier to create, deploy, and run applications by using containers
- Containers allow you to package up an application with all of the parts it needs, such as libraries and other dependencies, and ship it out as one package
- Docker is *a bit like* a virtual machine, but rather than creating a whole virtual operating system, Docker allows applications to use the same Linux kernel

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

Node.js 16 Docker Image

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

package.json

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

node_modules

package.json

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

Application

node_modules

package.json

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

Application

node_modules

package.json

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

Application

node_modules

package.json

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

Application

node_modules

package.json

Operating System Updates

Node.js 16 Docker Image

986 MB

Dockerfile

```
FROM node:16

# Change working directory
WORKDIR "/app"

# Update packages and install dependency packages for services
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install npm production packages
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

EXPOSE 3000

USER node

CMD ["npm", "start"]
```

node_modules

package.json

Operating System Updates

Node.js 16 Docker Image

Dockerfile

```
# Install the app dependencies in a full Node docker image
FROM node:16
WORKDIR "/app"

# Install OS updates
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install app dependencies
RUN npm install --production

# Copy the dependencies into a Slim Node docker image
FROM node:14-slim
 WORKDIR "/app"

# Install OS updates
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Install app dependencies
COPY --from=0 /app/node_modules /app/node_modules
COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

EXPOSE 3000
CMD ["npm", "start"]
```

node_modules

package.json

Operating System Updates

Node.js 16 Slim Docker Image

Dockerfile

```
# Install the app dependencies in a full Node docker image
FROM node:16
WORKDIR "/app"

# Install OS updates
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install app dependencies
RUN npm install --production

# Copy the dependencies into a Slim Node docker image
FROM node:16-slim
WORKDIR "/app"

# Install OS updates
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Install app dependencies
COPY --from=0 /app/node_modules /app/node_modules
COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

EXPOSE 3000
CMD ["npm", "start"]
```

Application

node_modules

package.json

Operating System Updates

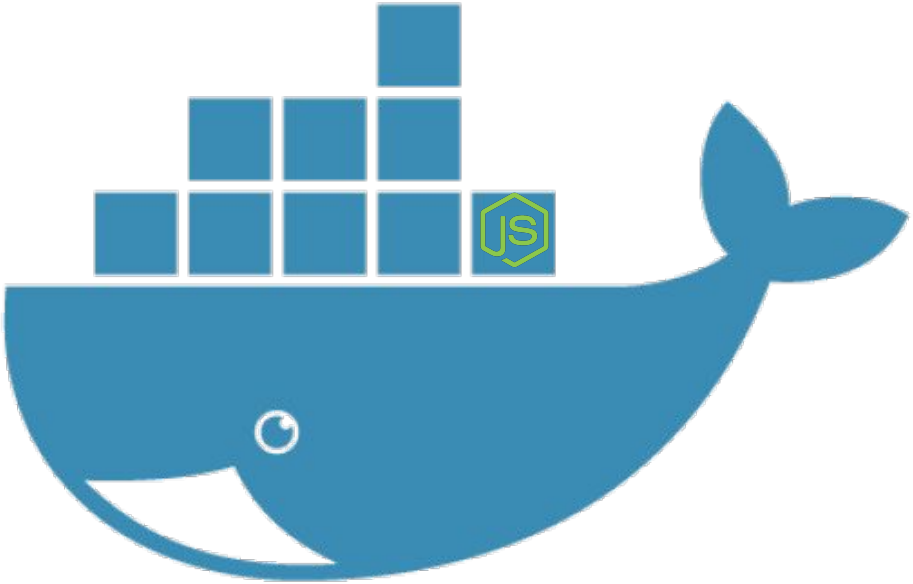
Node.js 16 Slim Docker Image

190 MB

Dockerfile

```
# Install the app dependencies in a full Node docker image
FROM node:16
WORKDIR "/app"

# Install OS updates
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Copy package.json and package-lock.json
COPY package*.json ./

# Install app dependencies
RUN npm install --production

# Copy the dependencies into a Slim Node docker image
FROM node:16-slim
WORKDIR "/app"

# Install OS updates
RUN apt-get update \
&& apt-get dist-upgrade -y \
&& apt-get clean \
&& echo 'Finished installing dependencies'

# Install app dependencies
COPY --from=0 /app/node_modules /app/node_modules
COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

EXPOSE 3000
CMD ["npm", "start"]
```


```
$ docker build --tag nodeserver --file Dockerfile-run .
$ docker run --detach --publish 3000:3000 --tty nodeserver
```



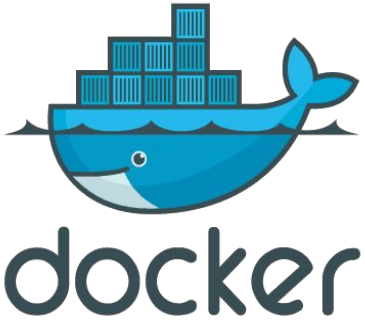
How to I build containers on Windows/Mac ?

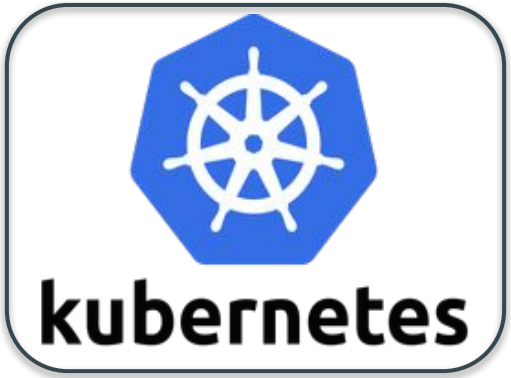
- Option 1 - Docker Desktop
  - Supports building containers on Windows/Mac
  - Unfortunately it’s no longer free - unless company is <250 people and < $10M
  - It’s not open source
- Option 2 - Podman
  - Free
  - OpenSource

- Build time
  - You can consume the the same base containers
  - You can use the same Docker/Container files
  - Many/most of the commands are the same
  - You end up with the container

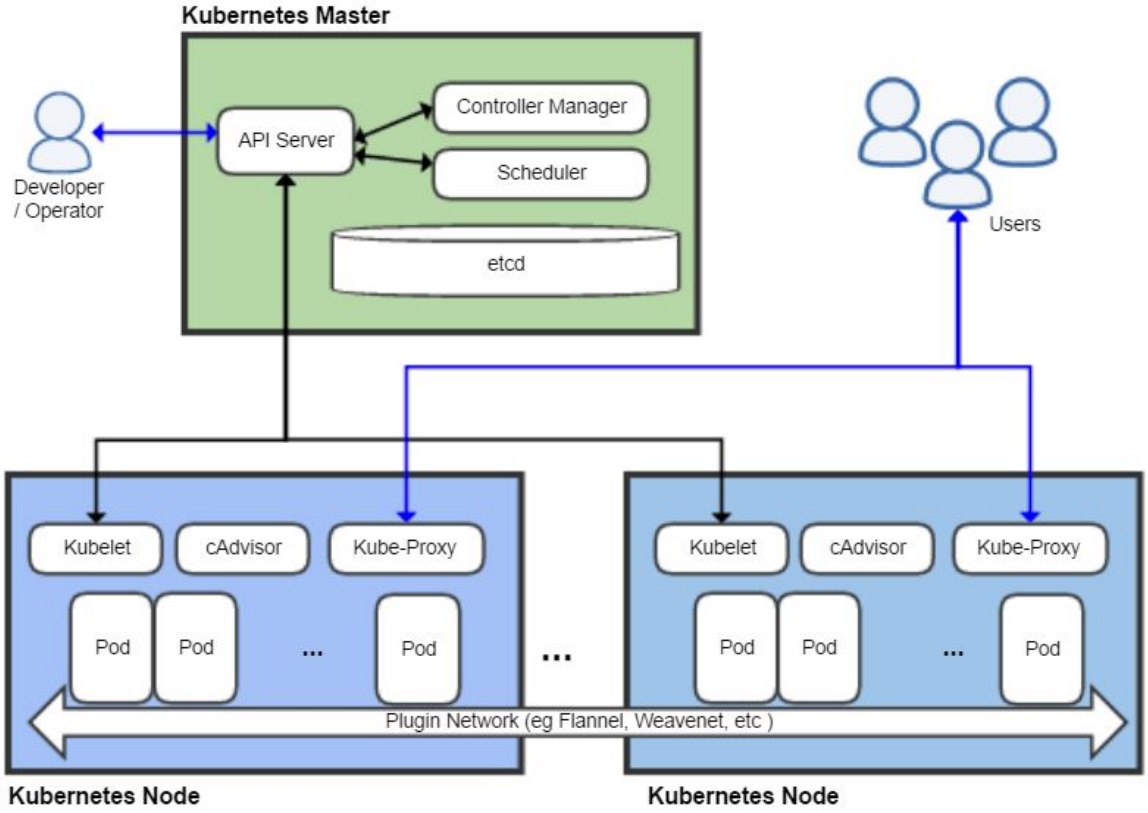
- Run time
  - Containers can on a number of different runtimes

CLOUD NATIVE
COMPUTING FOUNDATION





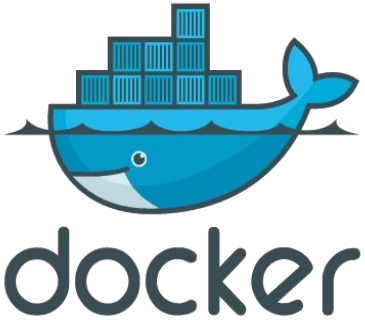
kubernetes

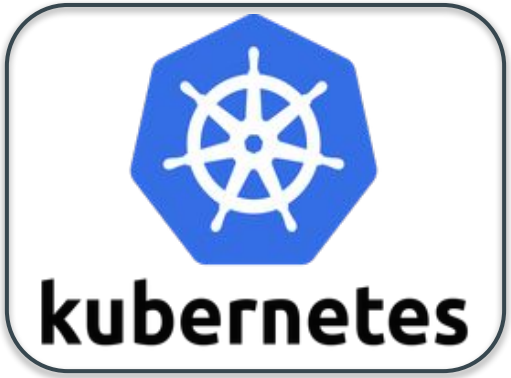

Kubernetes Master
Developer / Operator
API Server
Controller Manager
Scheduler
etcd
Users
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
...
Kubelet
cAdvisor
Kube-Proxy
Pod
Pod
...
Pod
Plugin Network (eg Flannel, Weavenet, etc)
Kubernetes Node
Kubernetes Node

Manages Your Containers

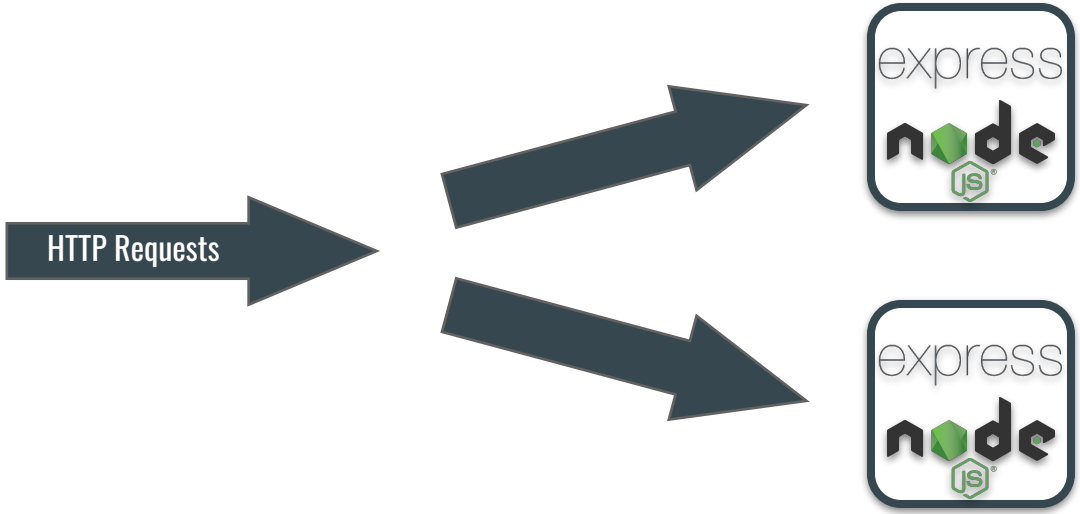
- Service discovery and load balancing
- Storage orchestration
- Automated rollouts and rollbacks
- Automatic bin packing - specifying how much CPU and memory each container needs
- Self-healing
- Secret and configuration management

CLOUD NATIVE
COMPUTING FOUNDATION

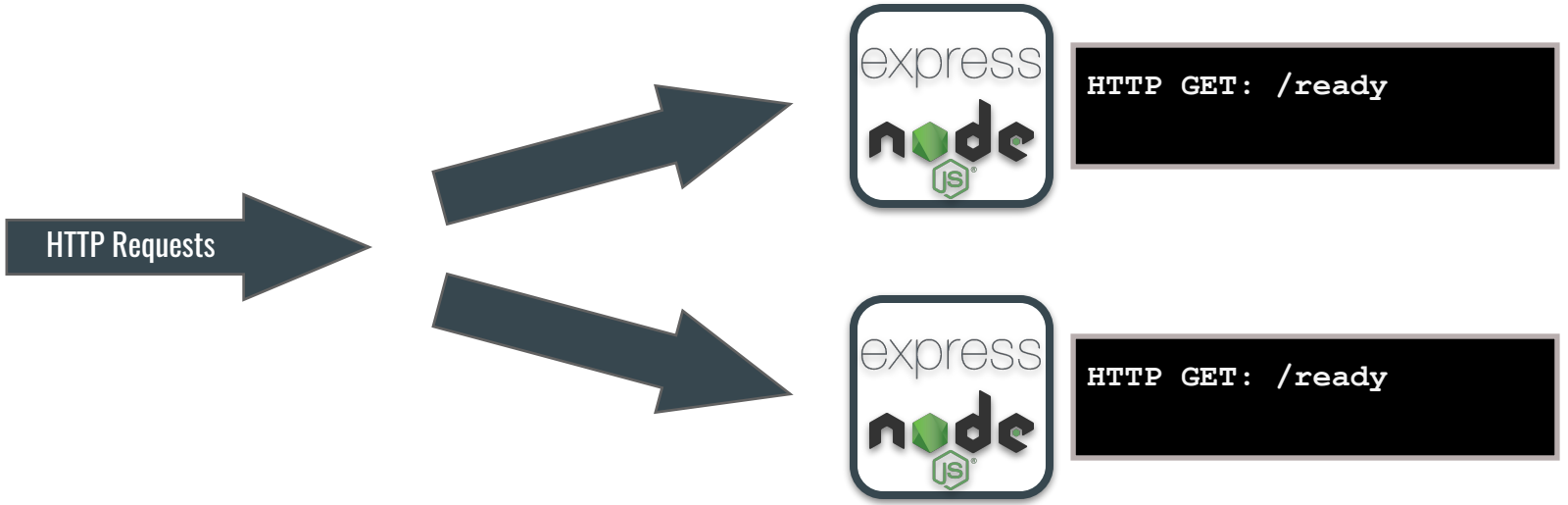




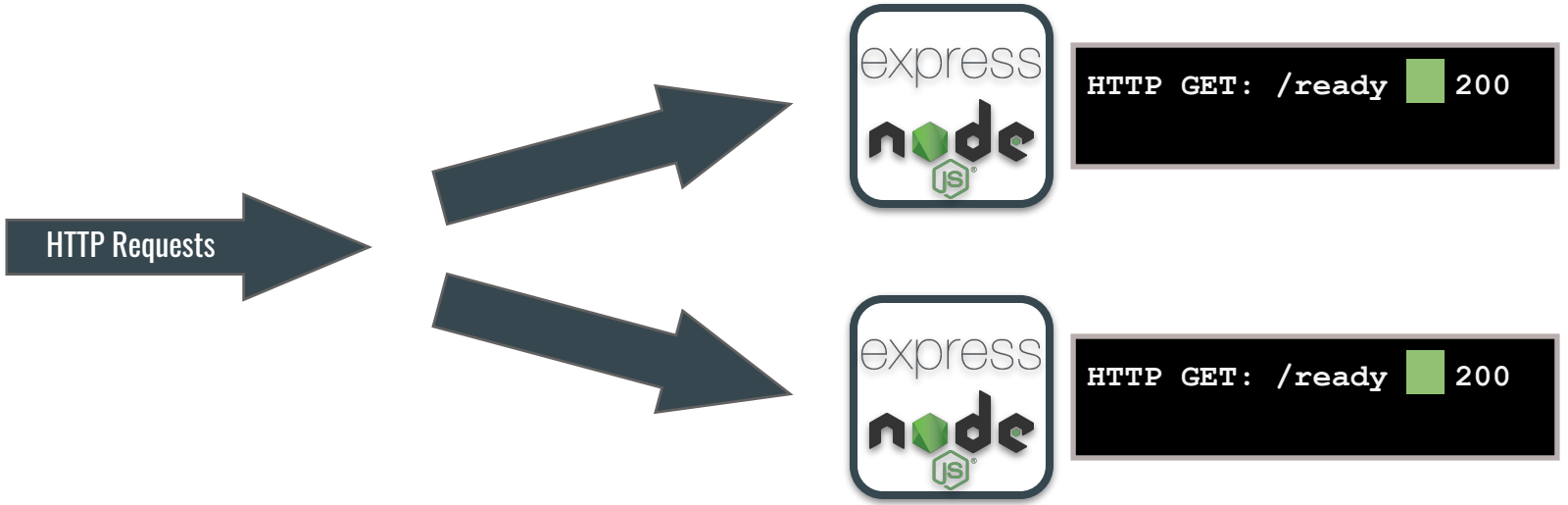
Health Checks



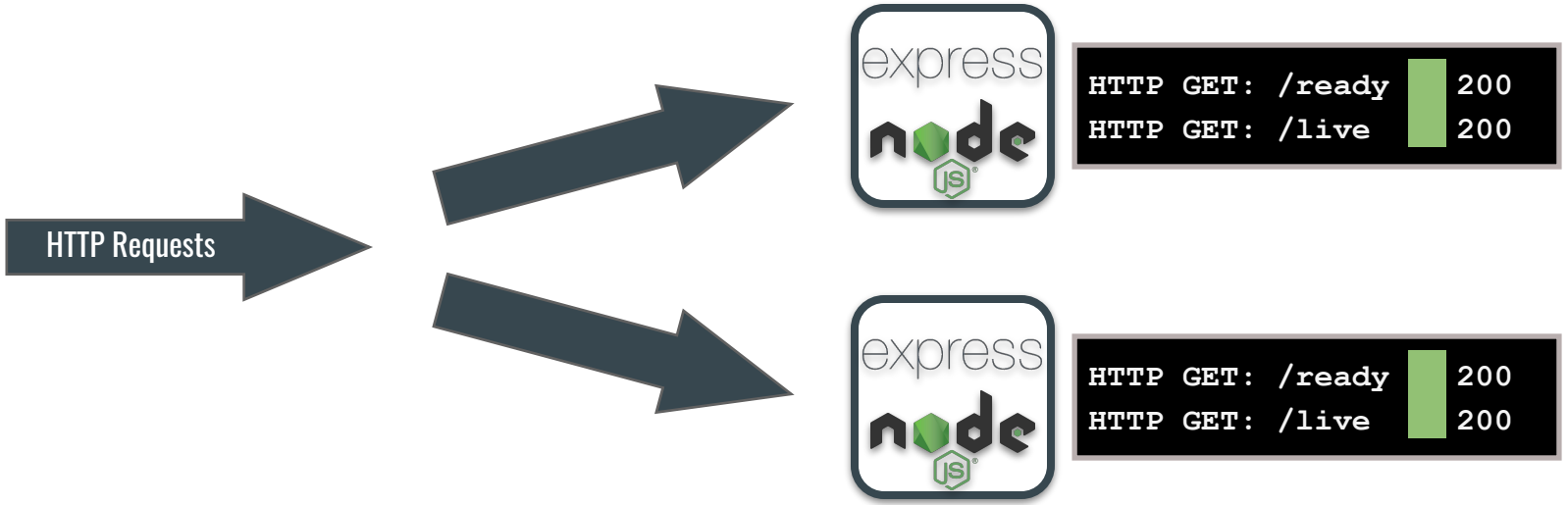
Health Checks



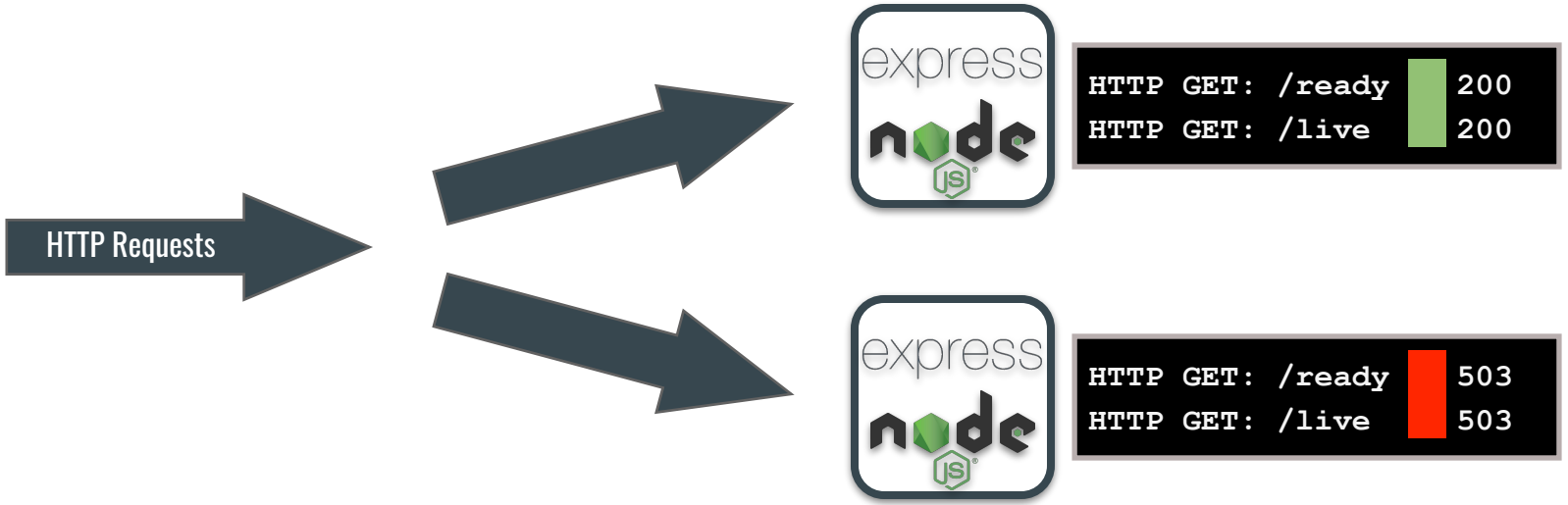
Health Checks



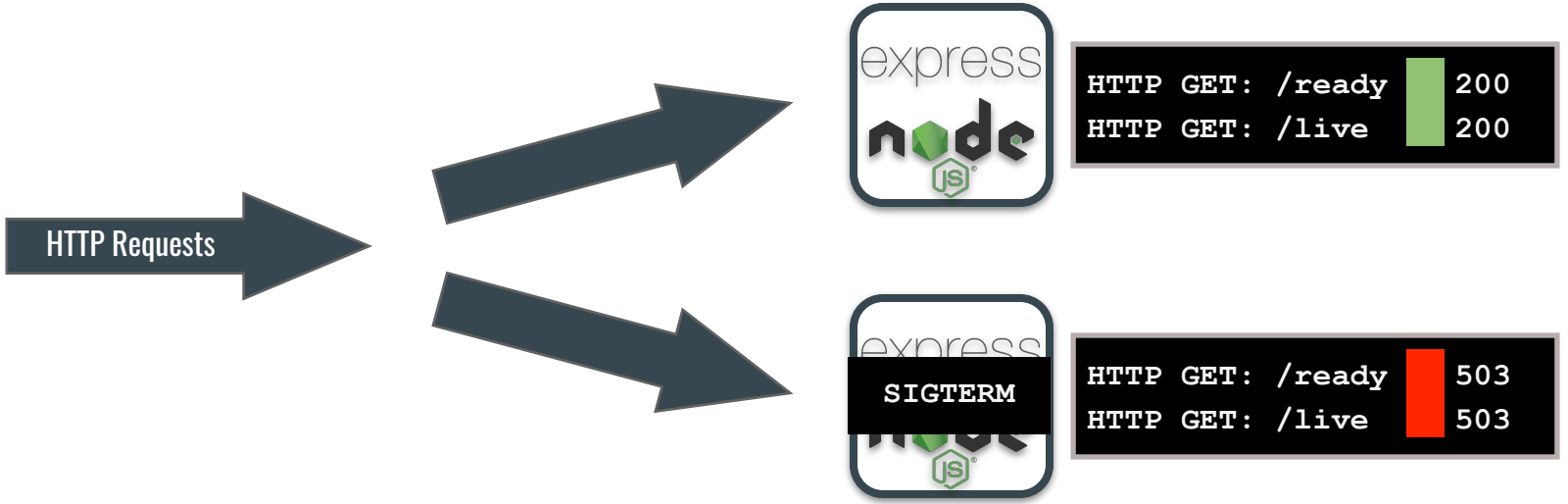
Health Checks



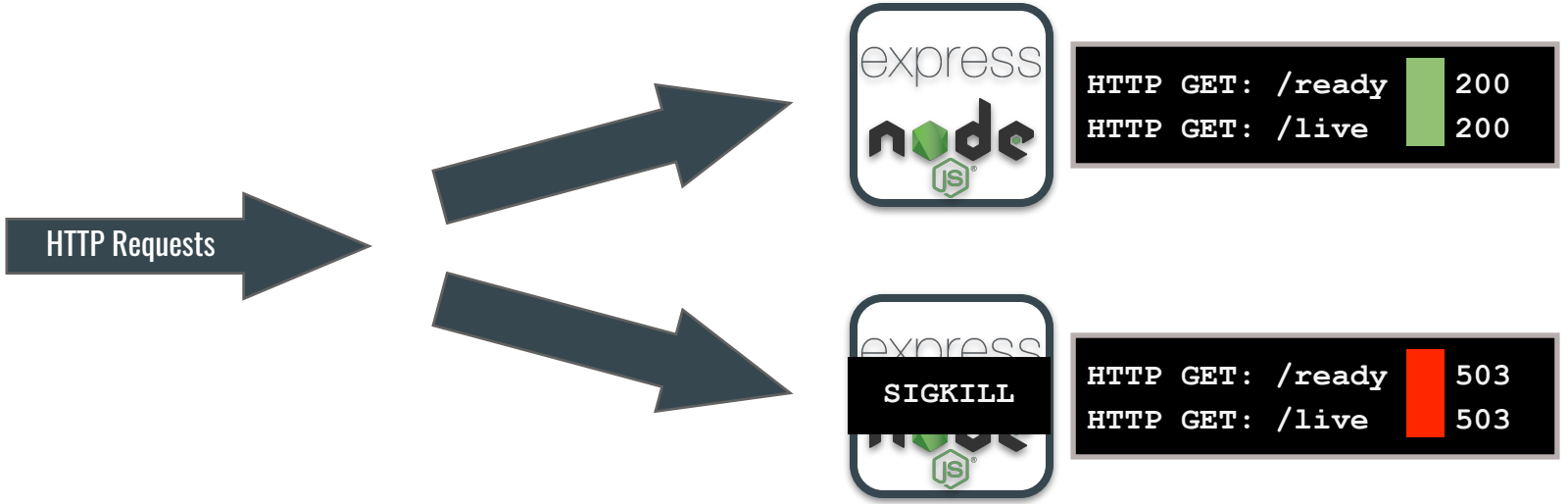
Health Checks



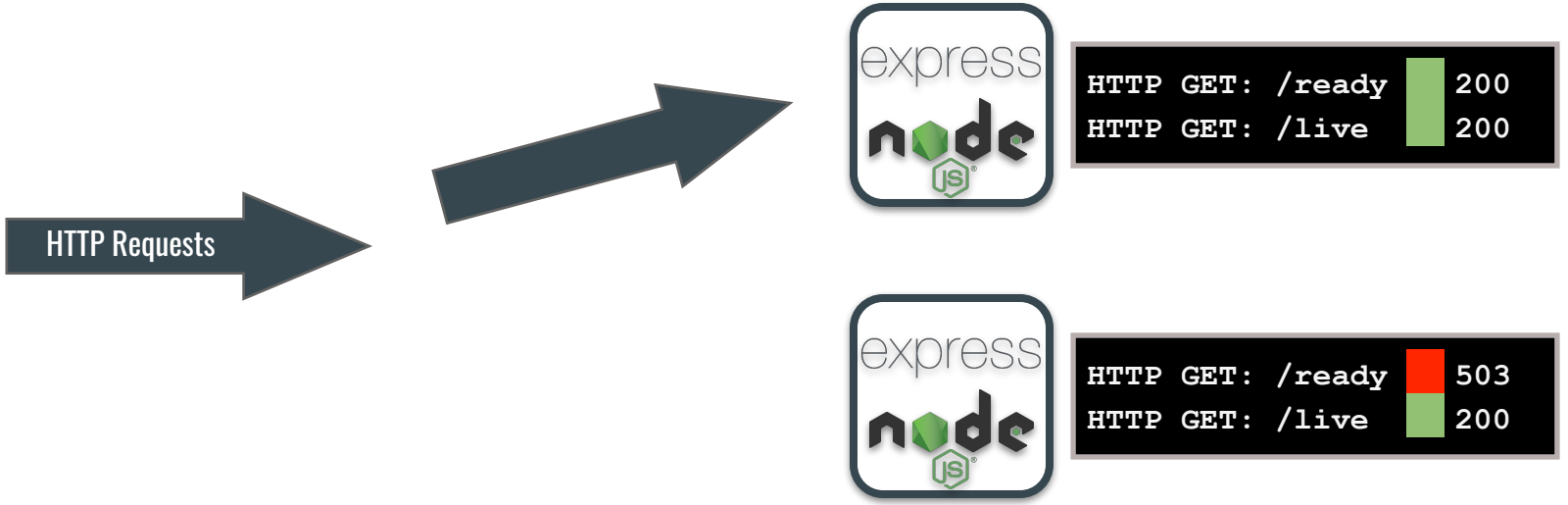
Health Checks



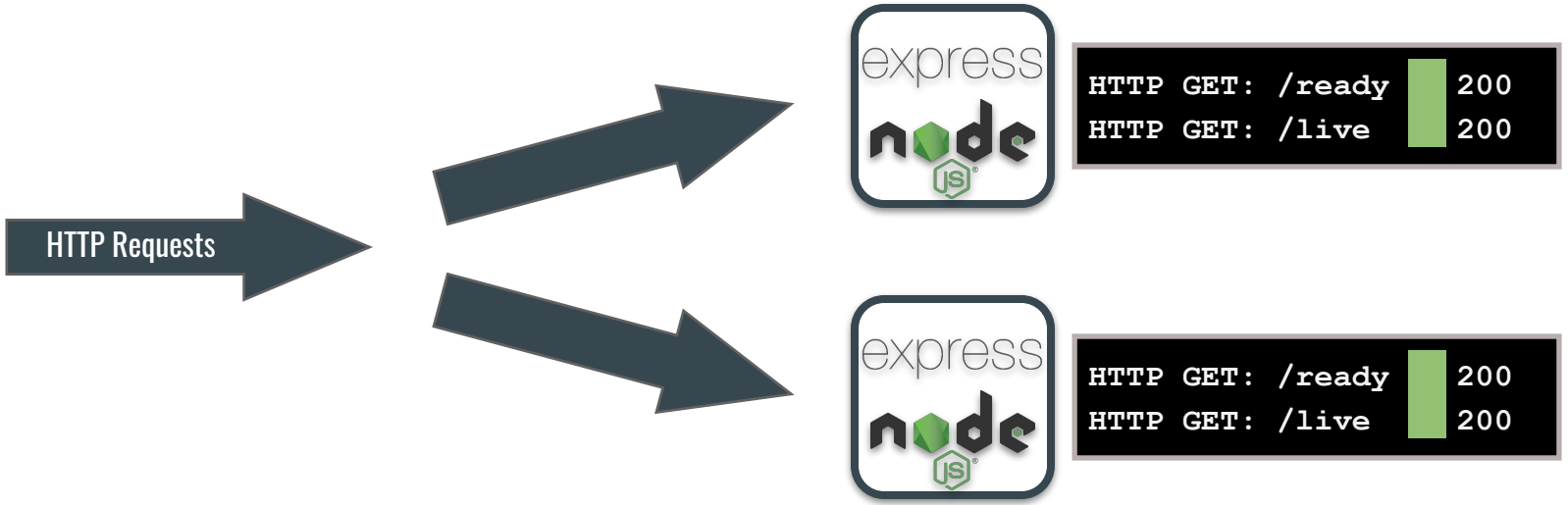
Health Checks



Health Checks



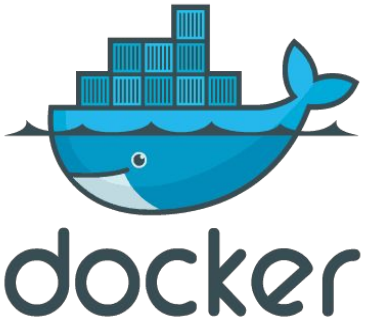
Health Checks



Health Checks

```
const app = require("express")();

// Note that when collecting metrics, the management endpoints should be
// implemented before the instrumentation that collects metrics, so that
// these endpoints are not counted in the metrics.
app.get("/ready", (req, res) => res.status(200).json({ status: "ok" }));
app.get("/live", (req, res) => res.status(200).json({ status: "ok" }));

// ... rest of app...

app.listen();
```

CLOUD NATIVE
COMPUTING FOUNDATION



kubernetes

Prometheus

HELM CHARTS

- Helm uses a packaging format called *charts*.
- A chart is a collection of files that describe a related set of Kubernetes resources.
- *A bit like* package.json for Kubernetes deployment

HELM CHARTS

```
$ helm create chart
$ ls
chart/
├── .helmignore   # Contains patterns to ignore when packaging Helm charts.
├── Chart.yaml    # Information about your chart
├── values.yaml   # The default values for your templates
├── charts/       # Charts that this chart depends on
└── templates/    # The template files
    └── tests/    # The test files
```

HELM CHARTS

```
$ helm create chart
$ ls
chart/
├── .helmignore
├── Chart.yaml
├── values.yaml
├── charts/
└── templates/
    └── tests/
```

```yaml
apiVersion: v2
name: node-app
description: A sample Helm chart
type: application
version: 0.1.0
appVersion: "1.16.0"
```

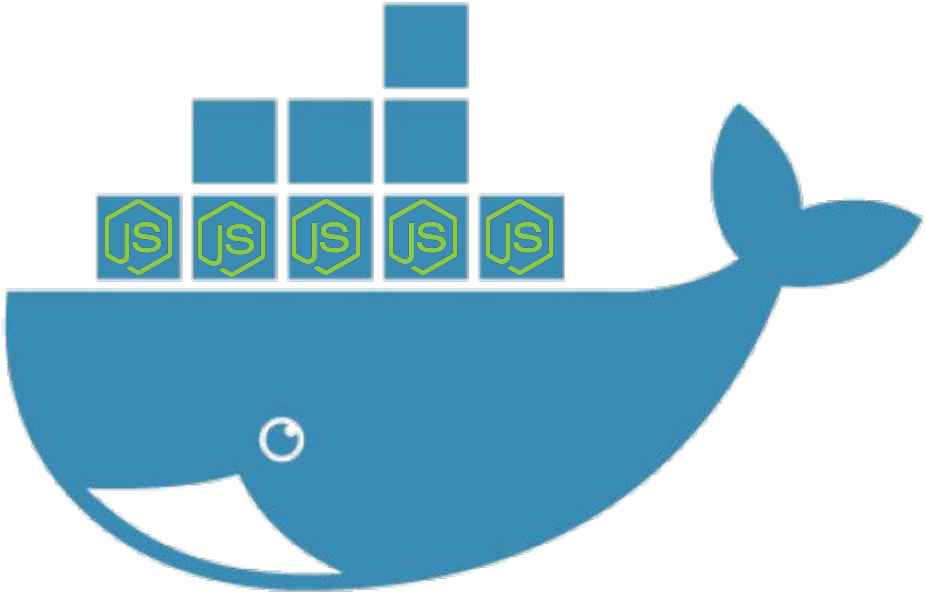
HELM CHARTS

```
$ helm create chart
$ ls
chart/
├── .helmignore
├── Chart.yaml
├── values.yaml
├── charts/
└── templates/
    └── tests/
```

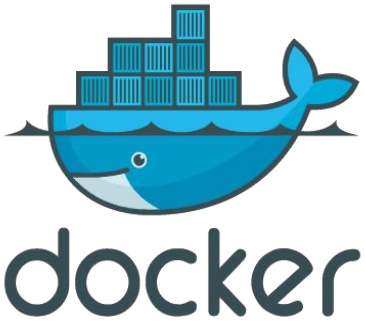
```yaml
replicaCount: 1
image:
 repository: nodeserver
 pullPolicy: IfNotPresent
 tag: ""
service:
 type: ClusterIP
 port: 80
resources:
 limits:
   cpu: 100m
   memory: 128Mi
autoscaling:
 enabled: false
 minReplicas: 1
 maxReplicas: 100
 targetCPUUtilizationPercentage: 80
```

kubernetes

```
$ cd ./chart/nodeserver/
$ helm install --name nodeserver .
```

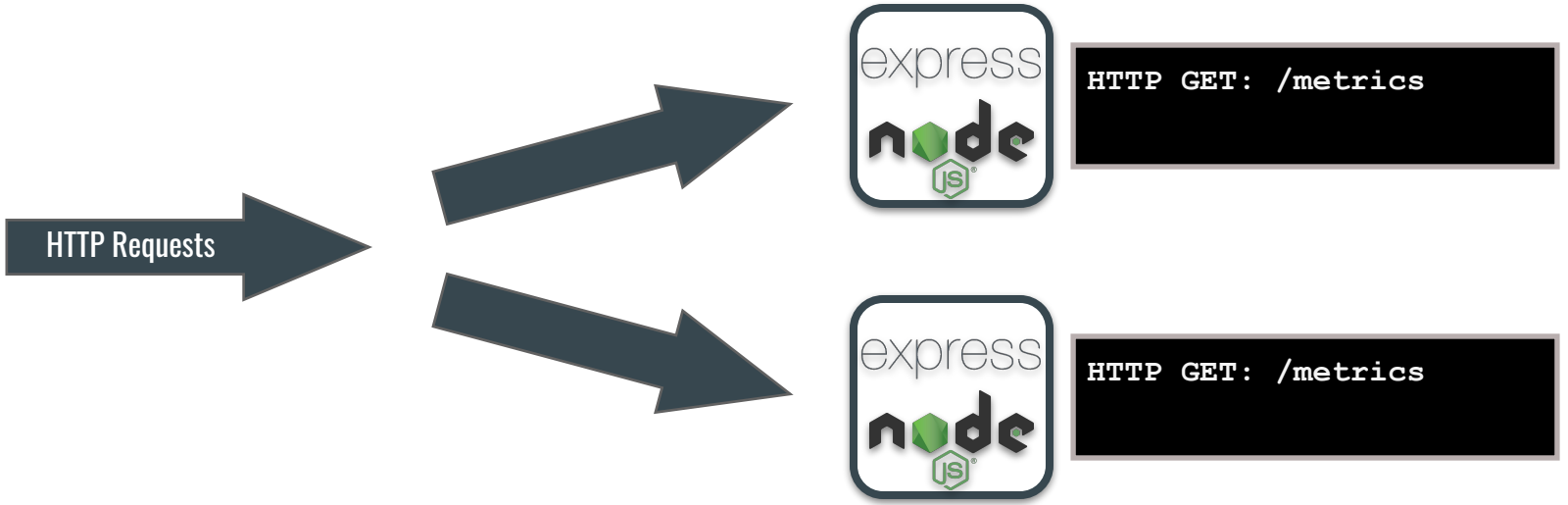


CLOUD NATIVE
COMPUTING FOUNDATION



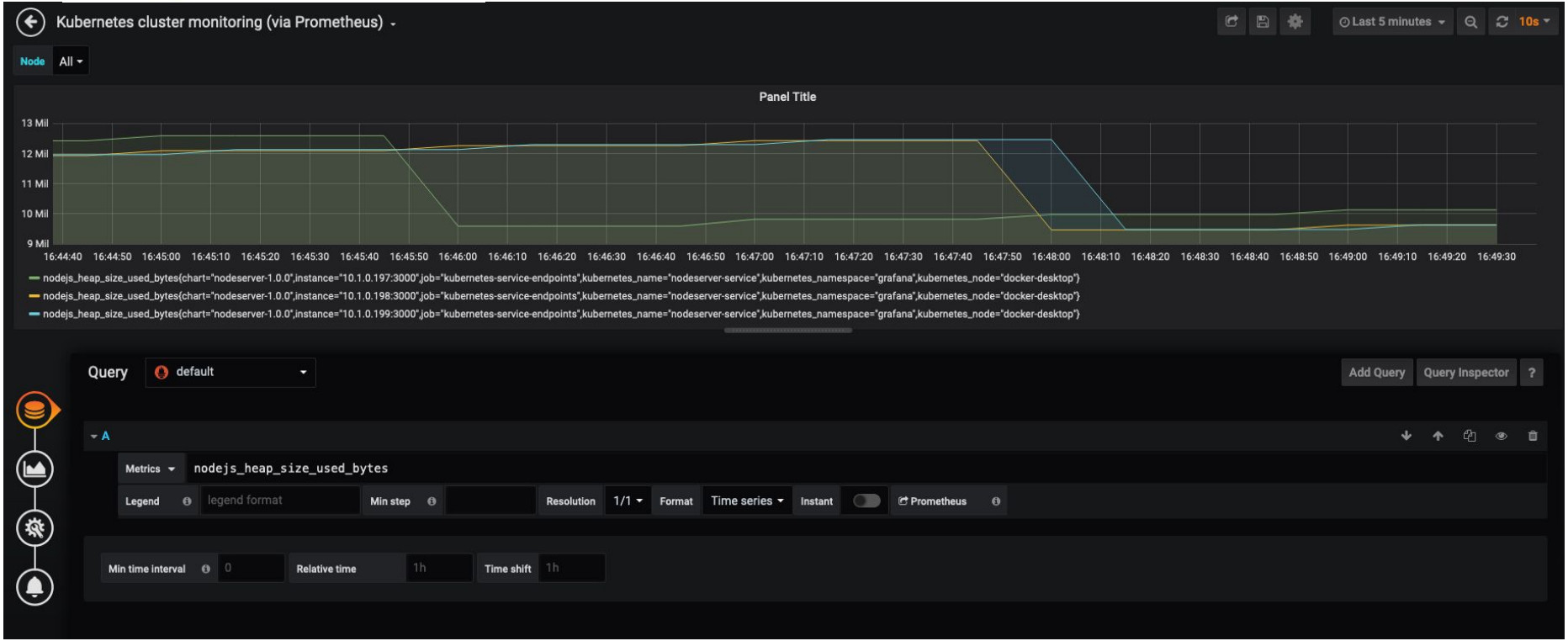
kubernetes

Prometheus


HTTP Requests
express
node
express
node
HTTP GET: /metrics
HTTP GET: /metrics


```
// Prometheus client setup
const Prometheus = require('prom-client');
Prometheus.collectDefaultMetrics();

app.get('/metrics', async (req, res, next) => {
  try {
     res.set('Content-Type', Prometheus.register.contentType);
     const metrics = await Prometheus.register.metrics();
     res.end(metrics);
  } catch {
     res.end('');
  }
});
```


```
# HELP process_cpu_user_seconds_total Total user CPU time spent in seconds.
# TYPE process_cpu_user_seconds_total counter
process_cpu_user_seconds_total 0.020806

# HELP process_cpu_system_seconds_total Total system CPU time spent in seconds.
# TYPE process_cpu_system_seconds_total counter
process_cpu_system_seconds_total 0.005973

# HELP process_cpu_seconds_total Total user and system CPU time spent in seconds.
# TYPE process_cpu_seconds_total counter
process_cpu_seconds_total 0.026779

# HELP process_start_time_seconds Start time of the process since unix epoch in seconds.
# TYPE process_start_time_seconds gauge
```


Kubernetes cluster monitoring (via Prometheus)
Last 5 minutes
10s
Node
All
Panel Title
13 Mil
12 Mil
11 Mil
10 Mil
9 Mil
16:44:40 16:44:50 16:45:00 16:45:10 16:45:20 16:45:30 16:45:40 16:45:50 16:46:00 16:46:10 16:46:20 16:46:30 16:46:40 16:46:50 16:47:00 16:47:10 16:47:20 16:47:30 16:47:40 16:47:50 16:48:00 16:48:10 16:48:20 16:48:30 16:48:40 16:48:50 16:49:00 16:49:10 16:49:20 16:49:30
nodejs_heap_size_used_bytes{chart="nodeserver-1.0.0",instance="10.1.0.197:3000",job="kubernetes-service-endpoints",kubernetes_name="nodeserver-service",kubernetes_namespace="grafana",kubernetes_node="docker-desktop"}
nodejs_heap_size_used_bytes{chart="nodeserver-1.0.0",instance="10.1.0.198:3000",job="kubernetes-service-endpoints",kubernetes_name="nodeserver-service",kubernetes_namespace="grafana",kubernetes_node="docker-desktop"}
nodejs_heap_size_used_bytes{chart="nodeserver-1.0.0",instance="10.1.0.199:3000",job="kubernetes-service-endpoints",kubernetes_name="nodeserver-service",kubernetes_namespace="grafana",kubernetes_node="docker-desktop"}
Query
default
Add Query
Query Inspector
A
Metrics
nodejs_heap_size_used_bytes
Legend
legend format
Min step
Resolution
1/1
Format
Time series
Instant
Prometheus
Min time interval
0
Relative time
1h
Time shift
1h

Summary

- Introduction to cloud-native development with Node.js.
- The workshop will cover key concepts and technologies, including:
  - Health checks
  - Metrics
  - Containers (Docker)
  - Kubernetes
  - Prometheus
  - Grafana

Node.js in the Cloud Workshop

🔗 Tutorial - https://red.ht/nodeconf2022