Think 2020

Lab: Cloud Native Development With Node.js

Agenda

- Intro to Cloud Native with Node.js
- Hands-on Lab
  - Part 1: Extending an Express.js Application to Leverage Cloud Capabilities
  - Part 2: Building Cloud-Native Apps with Application Stacks
  - Part 3: Introducing IBM Cloud Pak for Applications

CLOUD NATIVE ®
COMPUTING FOUNDATION

https://github.com/cncf

Docker®

https://www.docker.com

Prometheus ®

https://prometheus.io/

kubernetes®

https://github.com/kubernetes/kubernetes

HELM®

https://helm.sh/

CloudNativeJS.io

Docker

Kubernetes®

Health Checking

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

Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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

Docker examples are based on templates which are part of https://github.com/CloudNativeJS/docker (Apache License v2.0)

Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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

Node.js 10 Docker Image

Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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

Operating System Updates

Node.js 10 Docker Image

Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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

package.json

Operating System Updates

Node.js 10 Docker Image

Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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

node_modules

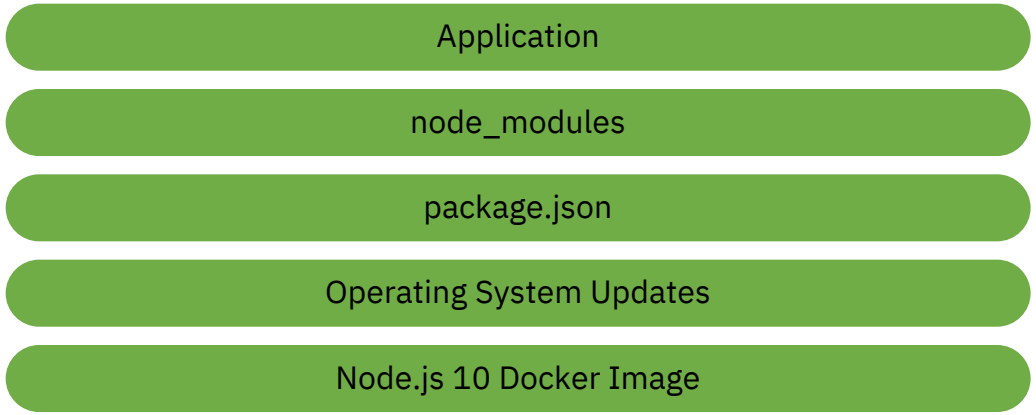
package.json

Operating System Updates

Node.js 10 Docker Image

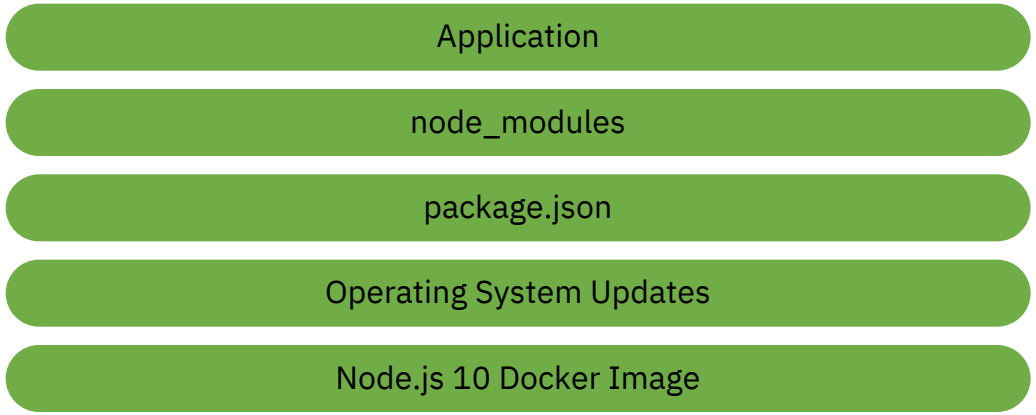
Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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



Docker

```
FROM node:10

# Change working directory
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 /app/

# Install app dependencies
RUN npm install --production

COPY . /app

ENV NODE_ENV production
ENV PORT 3000

USER node

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



716 MB

Docker

```
FROM node:10
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 package*.json /app/
RUN npm install --production
```

node_modules

package.json

Operating System Updates

Node.js 10 Docker Image

716 MB

Docker

```
FROM node:10
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 package*.json /app/
RUN npm install --production

# Copy the dependencies into a Slim Node docker image
FROM node:10-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 10 Docker Image

716 MB

Docker

```
FROM node:10
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 package.json /app/
RUN npm install –production

# Copy the dependencies into a Slim Node docker image
FROM node:10-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

716 MB

Docker

```
FROM node:10
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 package.json /app/
RUN npm install –production

# Copy the dependencies into a Slim Node docker image
FROM node:10-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

Node.js 10 **SLIM** Docker Image

716 MB

Docker

```
FROM node:8
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 package.json /app/
RUN npm install –production

# Copy the dependencies into a Slim Node docker image
FROM node:8-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

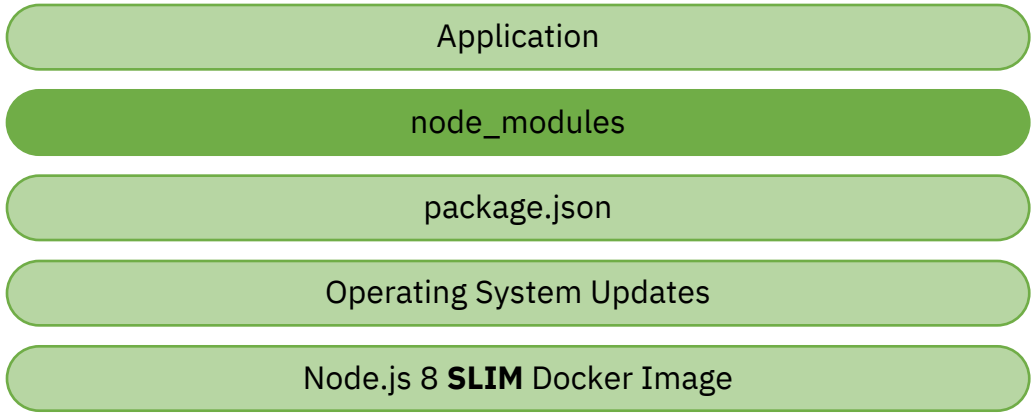
Operating System Updates

Node.js 8 **SLIM** Docker Image

716 MB

Docker

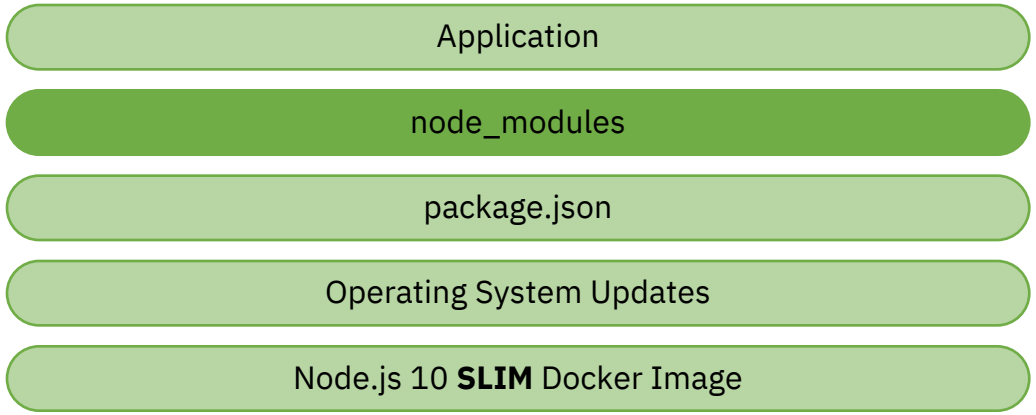
```
FROM node:8
WORKDIR "/app"

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

# Install app dependencies
COPY package.json /app/
RUN npm install –production

# Copy the dependencies into a Slim Node docker image
FROM node:8-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"]
```



716 MB

Docker

```
FROM node:10
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 package.json /app/
RUN npm install –production

# Copy the dependencies into a Slim Node docker image
FROM node:10-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"]
```



229 MB

Docker

```
$ docker build -t node-app -f Dockerfile-run .
$ docker run -d -p 3000:3000 -t node-app
```

CloudNativeJS.io

Docker

Kubernetes

Health Checking

Prometheus

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

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

Charts

```
/chart/nodeserver/templates/basedeployment.yaml
/chart/nodeserver/templates/deployment.yaml
/chart/nodeserver/templates/service.yaml
/chart/nodeserver/templates/hpa.yaml
/chart/nodeserver/templates/istio.yaml

/chart/nodeserver/Chart.yaml
/chart/nodeserver/values.yaml
```

```yaml
apiVersion: v1
description: A Helm chart for Kubernetes
name: nodeserver
version: 1.0.0
```

Helm examples are based on templates generated by the helm command line tool

Charts

```
/chart/nodeserver/templates/basedeployment.yaml
/chart/nodeserver/templates/deployment.yaml
/chart/nodeserver/templates/service.yaml
/chart/nodeserver/templates/hpa.yaml
/chart/nodeserver/templates/istio.yaml

/chart/nodeserver/Chart.yaml
/chart/nodeserver/values.yaml
```

```yaml
replicaCount: 1
revisionHistoryLimit: 1
image:
  repository: nodeserver
  tag: 1.0.0
  pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 200m
      memory: 300Mi
livenessProbe:
  initialDelaySeconds: 3000
  periodSeconds: 1000
service:
  name: Node
  type: NodePort
  servicePort: 3000
hpa:
  enabled: false
  minReplicas: 1
  maxReplicas: 2
  metrics:
    cpu:
      targetAverageUtilization: 70
    memory:
      targetAverageUtilization: 70
services:
base:
  enabled: false
  replicaCount: 1
  image:
    tag : v0.9.9
  weight: 100
istio:
  enabled: false
  weight: 100
```

Charts

```
/chart/nodeserver/templates/basedeployment.yaml
/chart/nodeserver/templates/deployment.yaml
/chart/nodeserver/templates/service.yaml
/chart/nodeserver/templates/hpa.yaml
/chart/nodeserver/templates/istio.yaml

/chart/nodeserver/Chart.yaml
/chart/nodeserver/values.yaml
```

```yaml
replicaCount: 1
revisionHistoryLimit: 1
image:
  repository: nodeserver
  tag: 1.0.0
  pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 200m
      memory: 300Mi
livenessProbe:
  initialDelaySeconds: 3000
  periodSeconds: 1000
service:
  name: Node
  type: NodePort
  servicePort: 3000
hpa:
  enabled: false
  minReplicas: 1
  maxReplicas: 2
  metrics:
    cpu:
      targetAverageUtilization: 70
    memory:
      targetAverageUtilization: 70
services:
base:
  enabled: false
  replicaCount: 1
  image:
    tag : v0.9.9
  weight: 100
istio:
  enabled: false
  weight: 100
```

Charts

```
/chart/nodeserver/templates/basedeployment.yaml
/chart/nodeserver/templates/deployment.yaml
/chart/nodeserver/templates/service.yaml
/chart/nodeserver/templates/hpa.yaml
/chart/nodeserver/templates/istio.yaml

/chart/nodeserver/Chart.yaml
/chart/nodeserver/values.yaml
```

```yaml
replicaCount: 1
revisionHistoryLimit: 1
image:
  repository: nodeserver
  tag: 1.0.0
  pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 200m
      memory: 300Mi
livenessProbe:
  initialDelaySeconds: 3000
  periodSeconds: 1000
service:
  name: Node
  type: NodePort
  servicePort: 3000
hpa:
  enabled: false
  minReplicas: 1
  maxReplicas: 2
  metrics:
    cpu:
      targetAverageUtilization: 70
    memory:
      targetAverageUtilization: 70
services:
base:
  enabled: false
  replicaCount: 1
  image:
    tag : v0.9.9
  weight: 100
istio:
  enabled: false
  weight: 100
```

Charts

```
/chart/nodeserver/templates/basedeployment.yaml
/chart/nodeserver/templates/deployment.yaml
/chart/nodeserver/templates/service.yaml
/chart/nodeserver/templates/hpa.yaml
/chart/nodeserver/templates/istio.yaml

/chart/nodeserver/Chart.yaml
/chart/nodeserver/values.yaml
```

```yaml
replicaCount: 1
revisionHistoryLimit: 1
image:
  repository: nodeserver
  tag: 1.0.0
  pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 200m
      memory: 300Mi
livenessProbe:
  initialDelaySeconds: 3000
  periodSeconds: 1000
service:
  name: Node
  type: NodePort
  servicePort: 3000
hpa:
  enabled: false
  minReplicas: 1
  maxReplicas: 2
  metrics:
    cpu:
      targetAverageUtilization: 70
    memory:
      targetAverageUtilization: 70
services:
base:
  enabled: false
  replicaCount: 1
  image:
    tag : v0.9.9
  weight: 100
istio:
  enabled: false
  weight: 100
```

Charts

```
/chart/nodeserver/templates/basedeployment.yaml
/chart/nodeserver/templates/deployment.yaml
/chart/nodeserver/templates/service.yaml
/chart/nodeserver/templates/hpa.yaml
/chart/nodeserver/templates/istio.yaml

/chart/nodeserver/Chart.yaml
/chart/nodeserver/values.yaml
```

```yaml
replicaCount: 1
revisionHistoryLimit: 1
image:
  repository: nodeserver
  tag: 1.0.0
  pullPolicy: IfNotPresent
  resources:
    requests:
      cpu: 200m
      memory: 300Mi
livenessProbe:
  initialDelaySeconds: 3000
  periodSeconds: 1000
service:
  name: Node
  type: NodePort
  servicePort: 3000
hpa:
  enabled: false
  minReplicas: 5
  maxReplicas: 9
  metrics:
    cpu:
      targetAverageUtilization: 70
    memory:
      targetAverageUtilization: 70
services:
base:
  enabled: false
  replicaCount: 1
  image:
    tag : v0.9.9
  weight: 100
istio:
  enabled: false
  weight: 100
```

kubernetes

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

CloudNativeJS.io

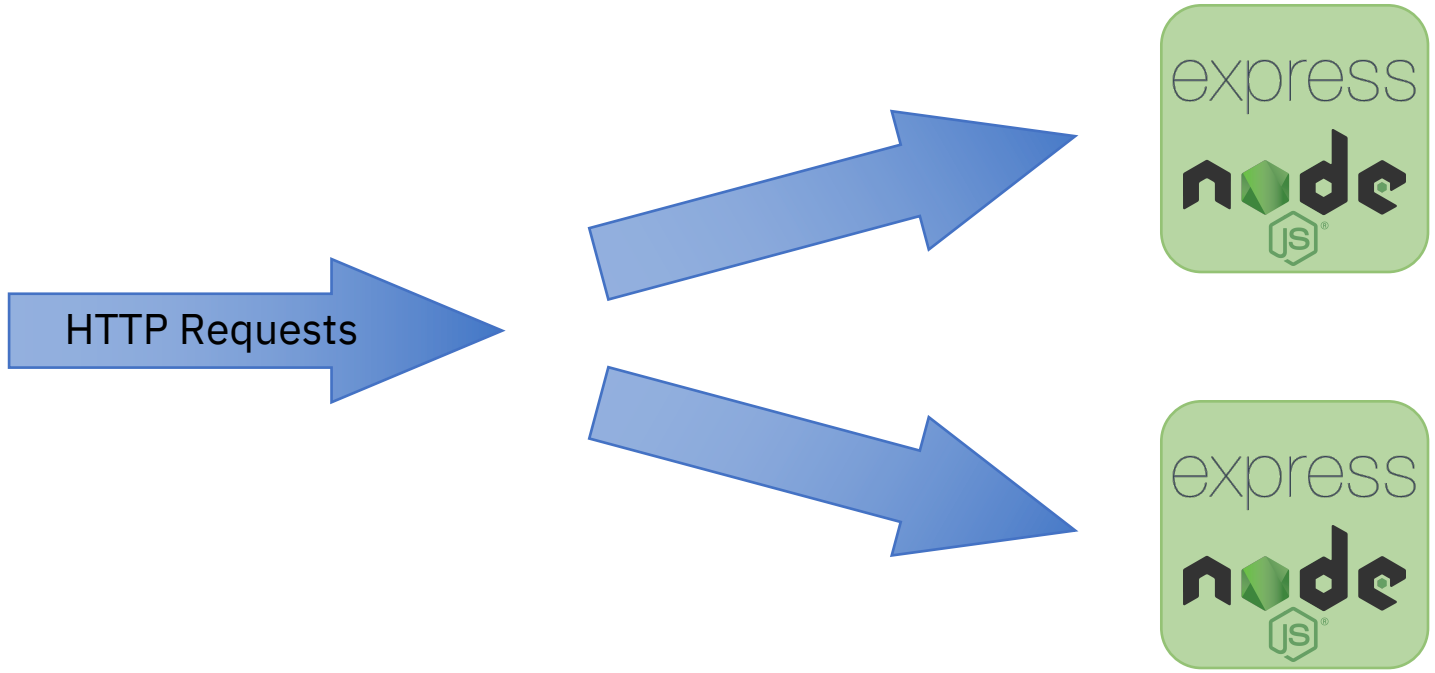
Docker

Kubernetes

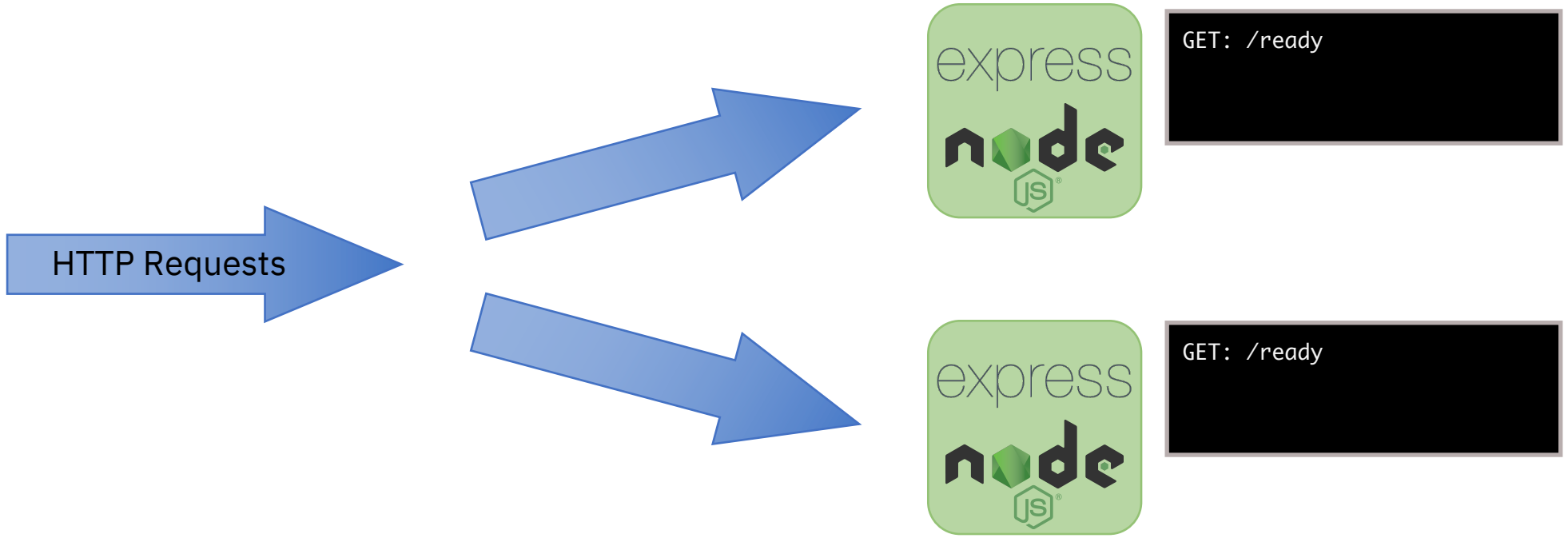
Health Checking

Prometheus

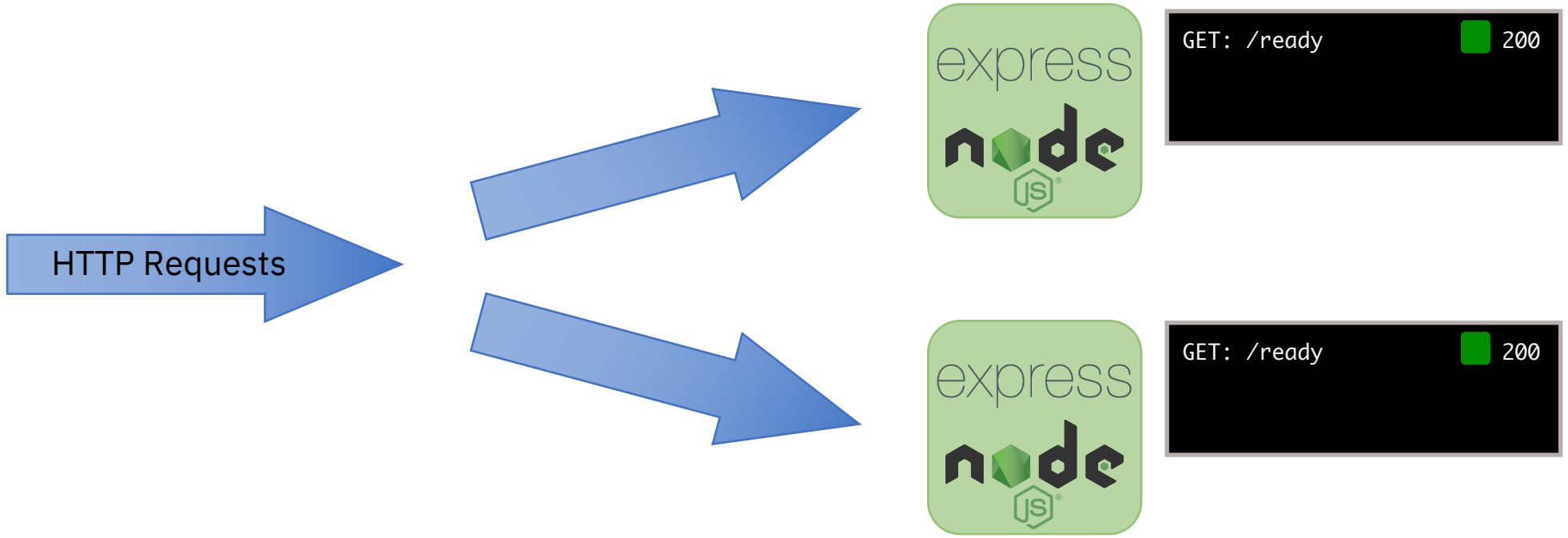
Health Checks



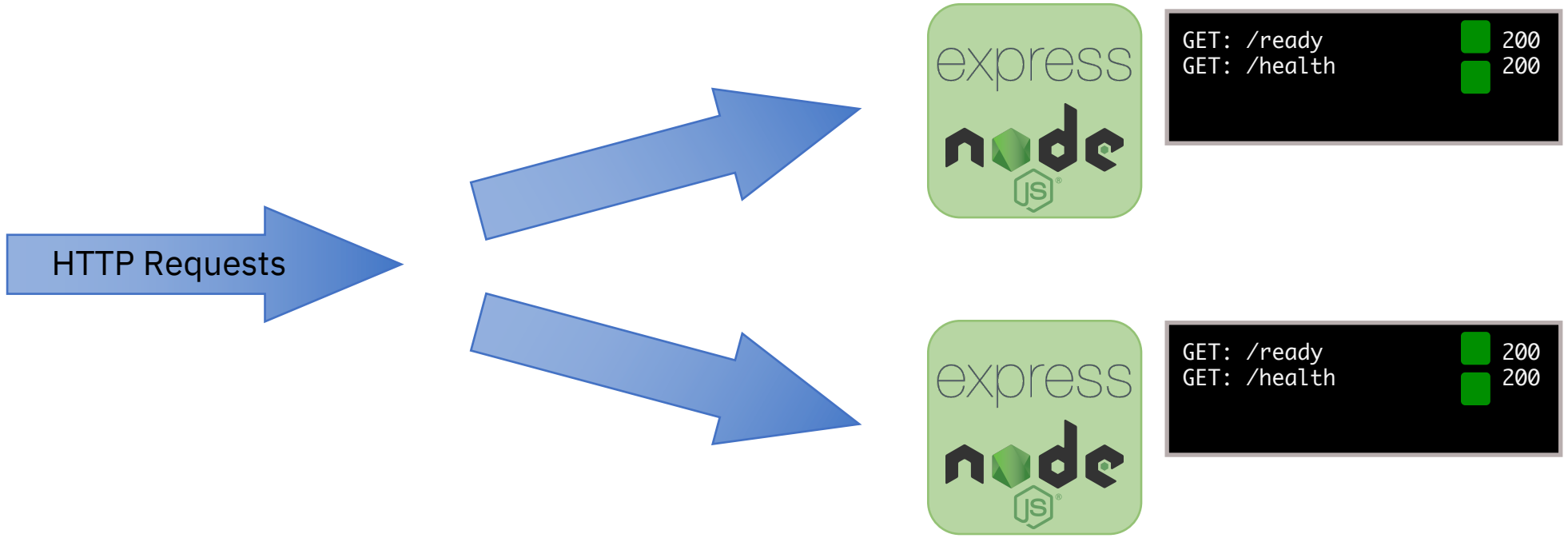
Health Checks



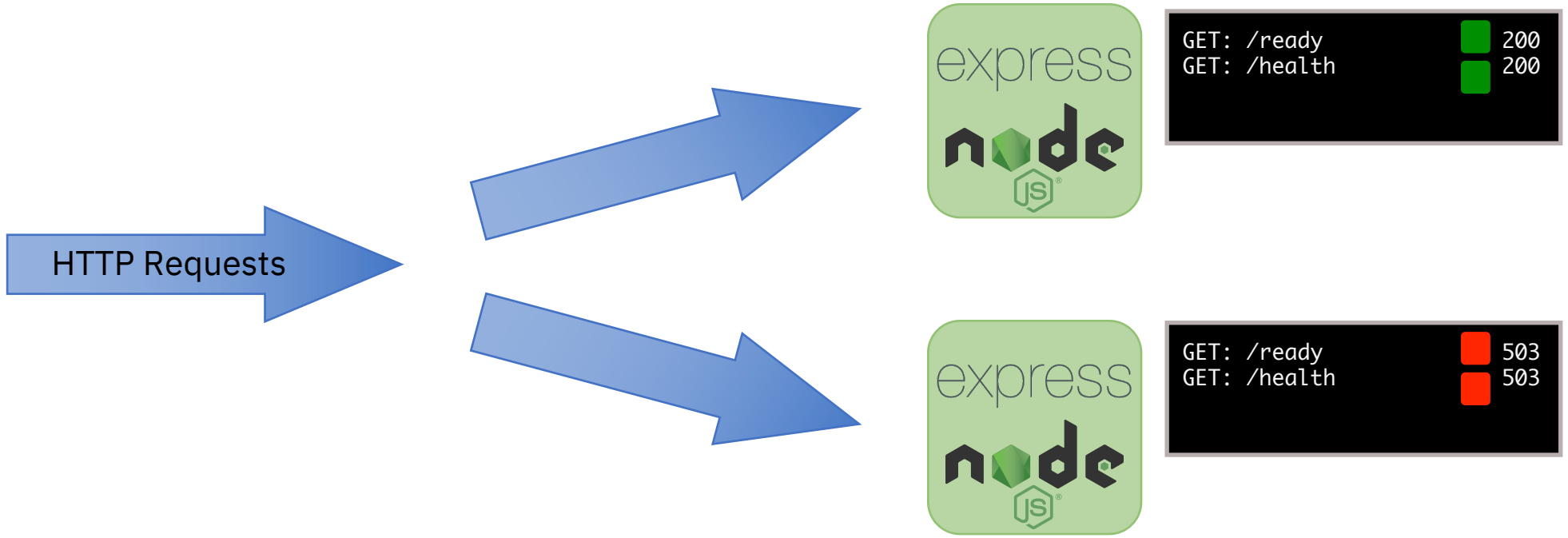
Health Checks



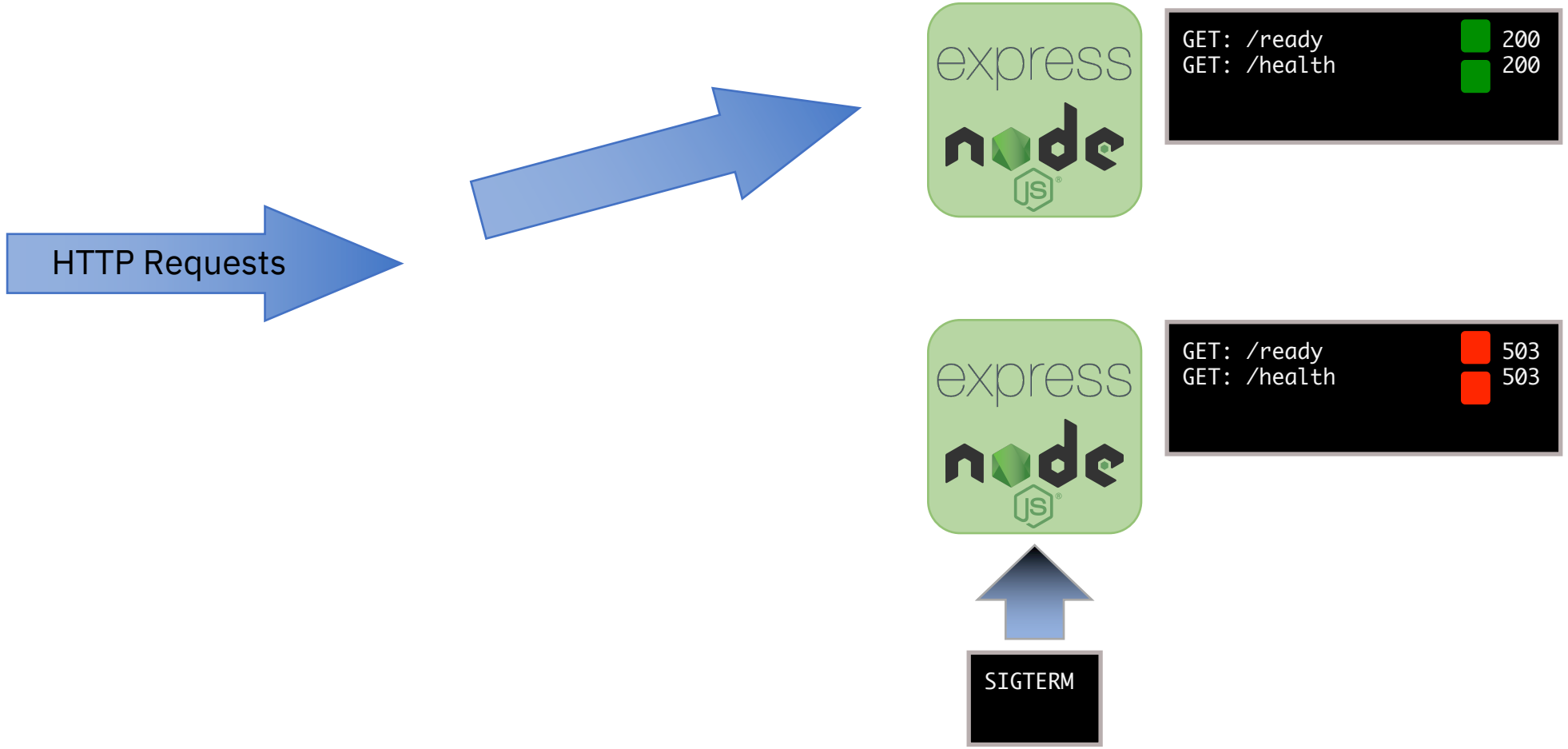
Health Checks



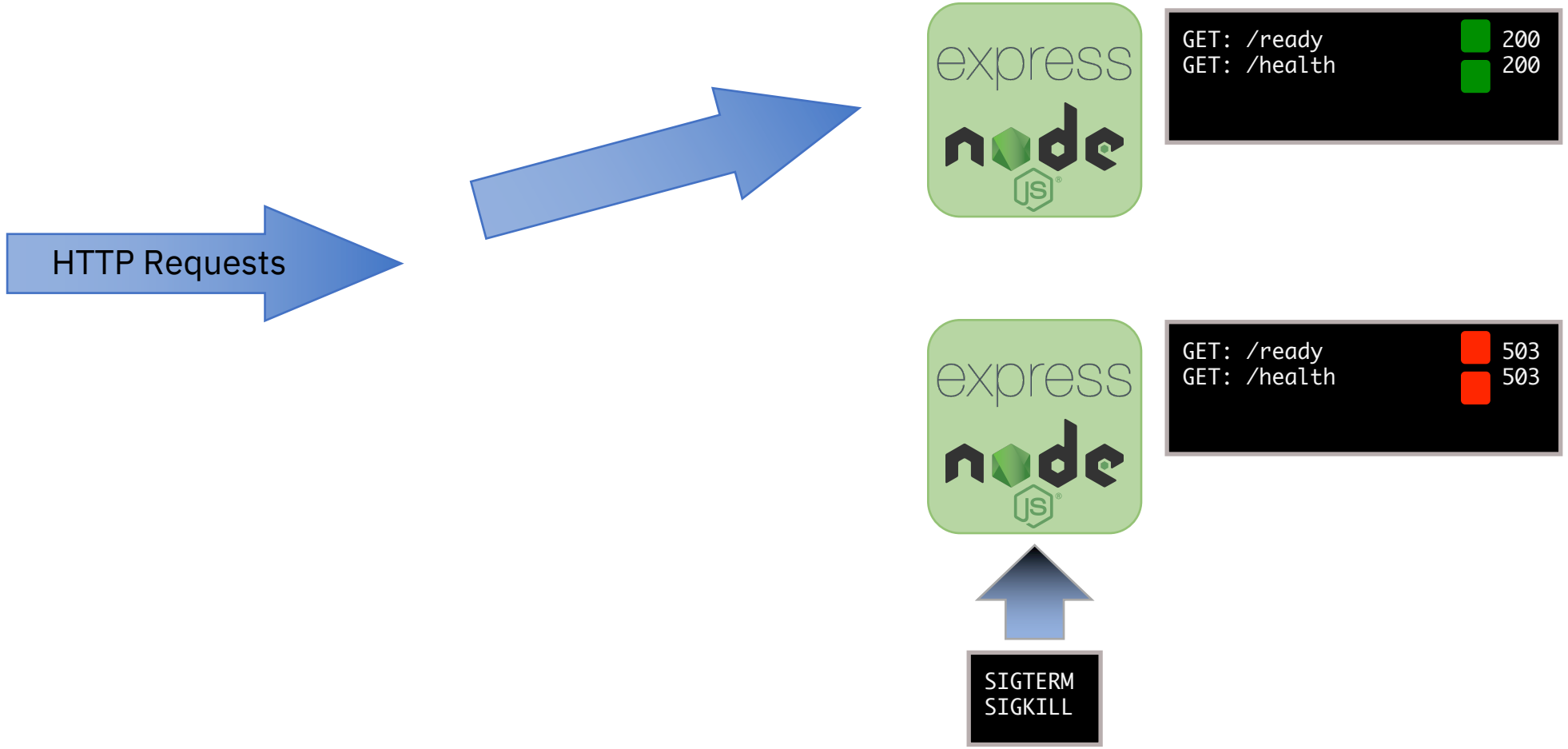
Health Checks



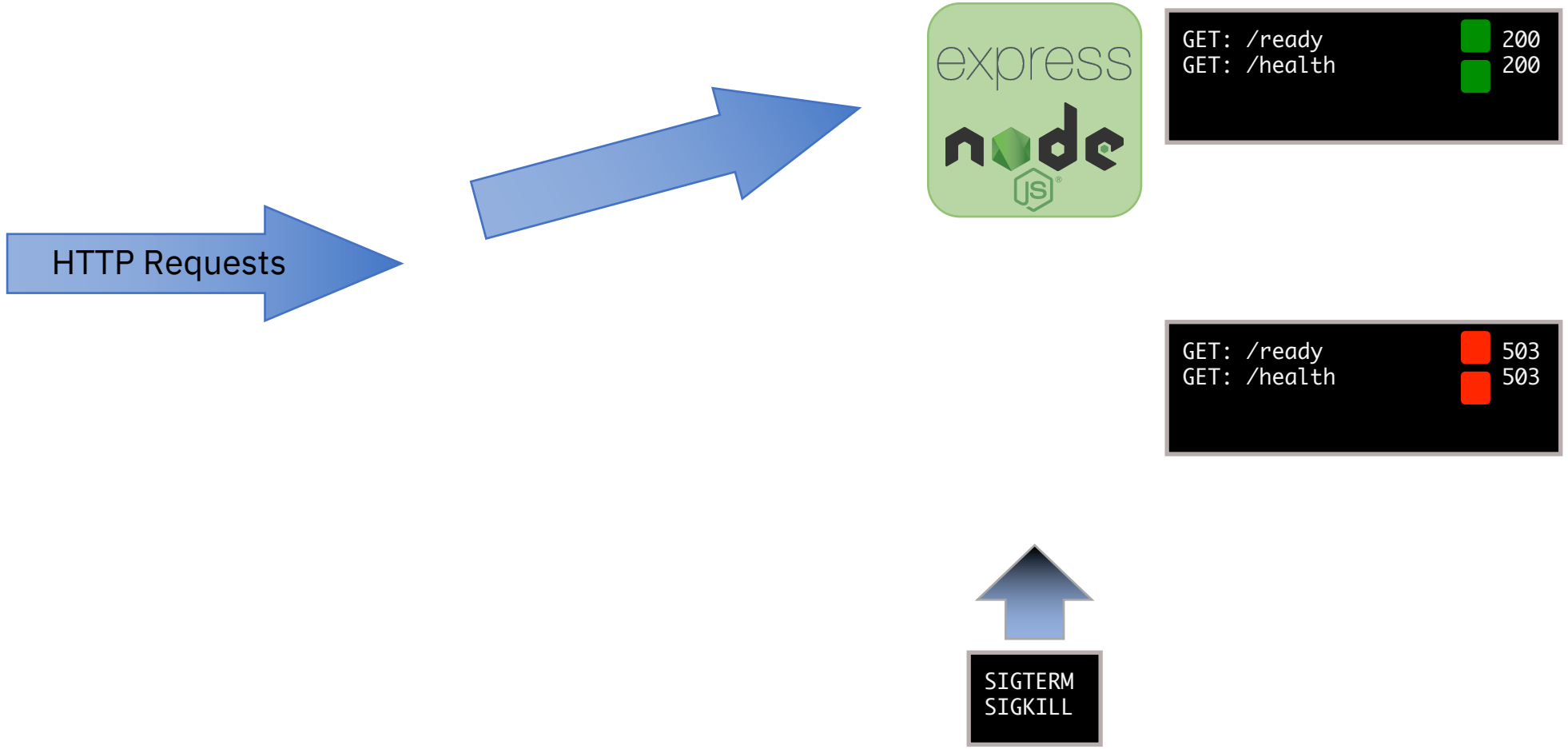
Health Checks



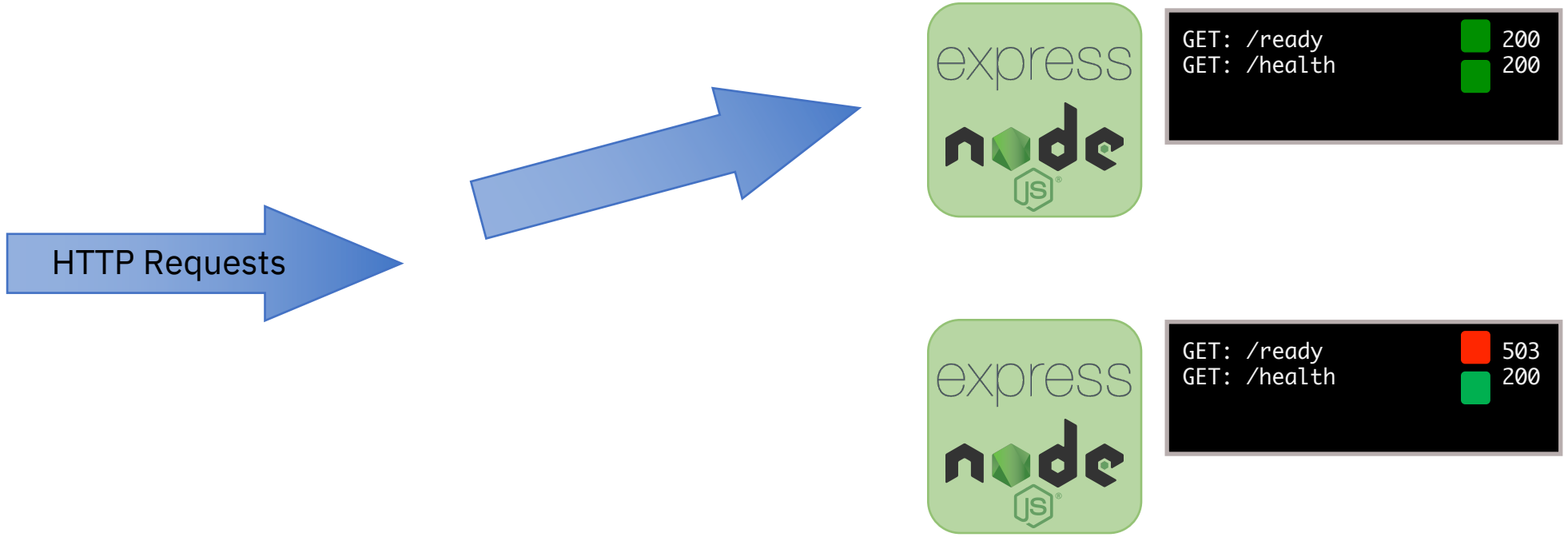
Health Checks



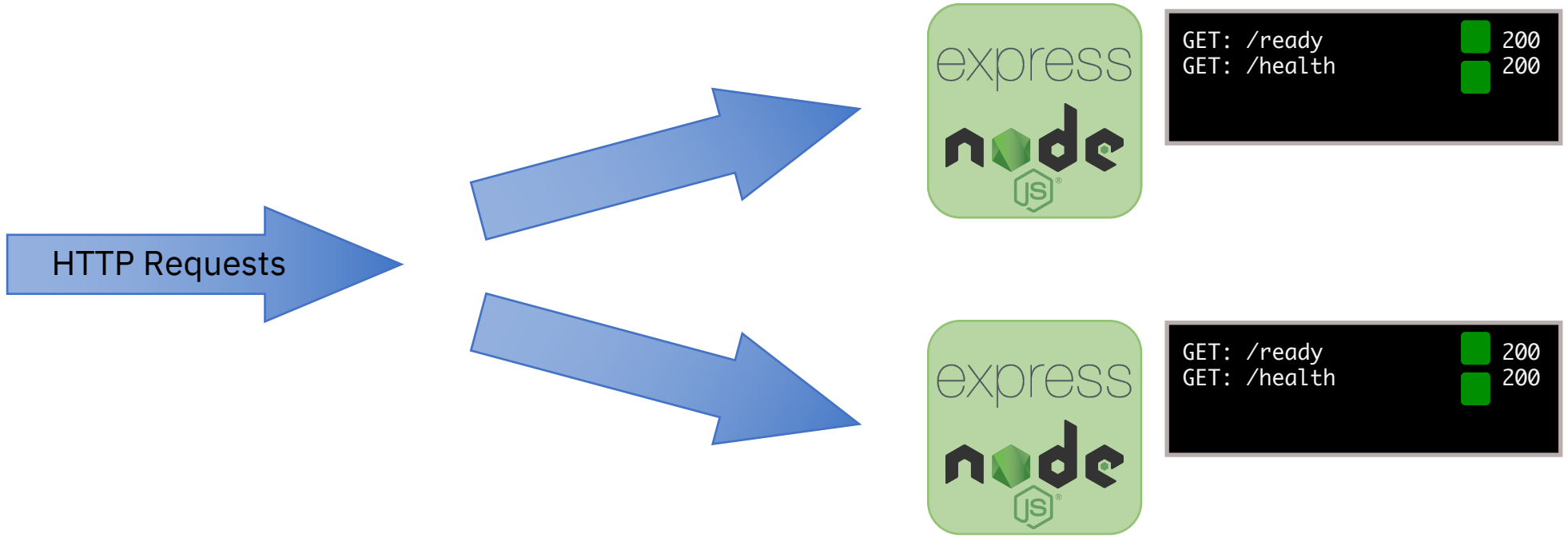
Health Checks



Health Checks



Health Checks



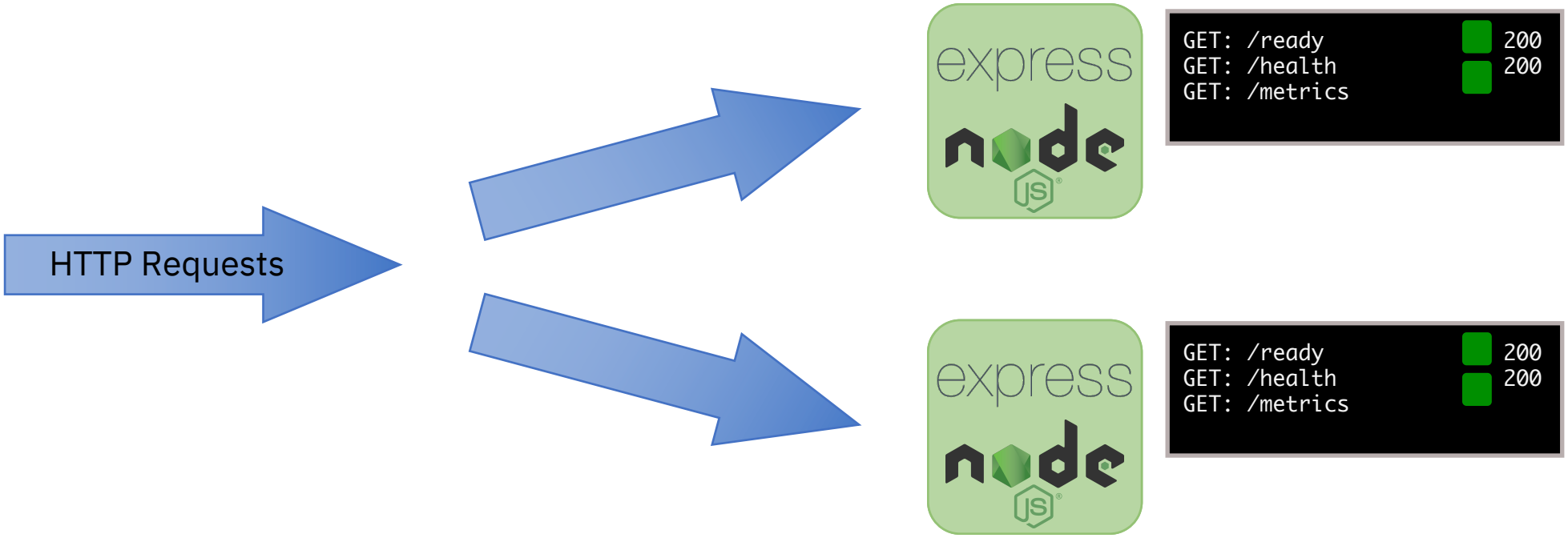
Health Checks

```
const health = require('@cloudnative/health-connect');
let healthcheck = new health.HealthChecker();

const readyPromise = new Promise(function (resolve, _reject) {
    resolve();
});
let readyCheck = new health.ReadinessCheck("ready", readyPromise);

const livePromise = new Promise(function (resolve, _reject) {
    resolve();
});
let liveCheck = new health.LivenessCheck("live", livePromise);

const shutdownPromise = new Promise(function (resolve, _reject) {
    resolve();
});
let shutdownCheck = new health.ShutdownCheck("shut", shutdownProm);

healthcheck.registerReadinessCheck(readyCheck);
healthcheck.registerLivenessCheck(liveCheck);
healthcheck.registerShutdownCheck(shutdownCheck);

app.use('/ready', health.ReadinessEndpoint(healthcheck))
app.use('/health', health.LivenessEndpoint(healthcheck))
```

CloudNativeJS.io

Docker
Kubernetes
Health Checking
Prometheus

Prometheus


HTTP Requests
express
node
GET: /ready 200
GET: /health 200
GET: /metrics
express
node
GET: /ready 200
GET: /health 200
GET: /metrics

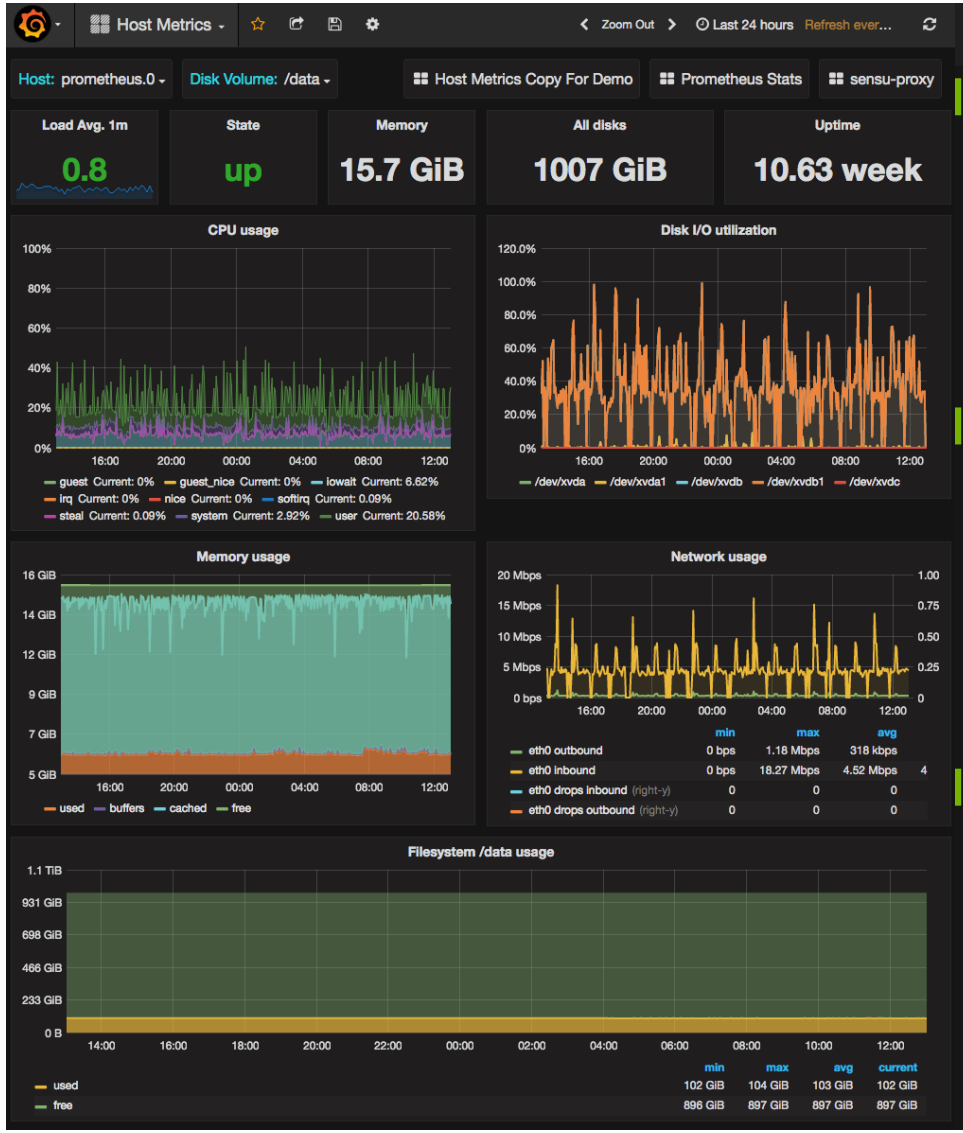
Prometheus

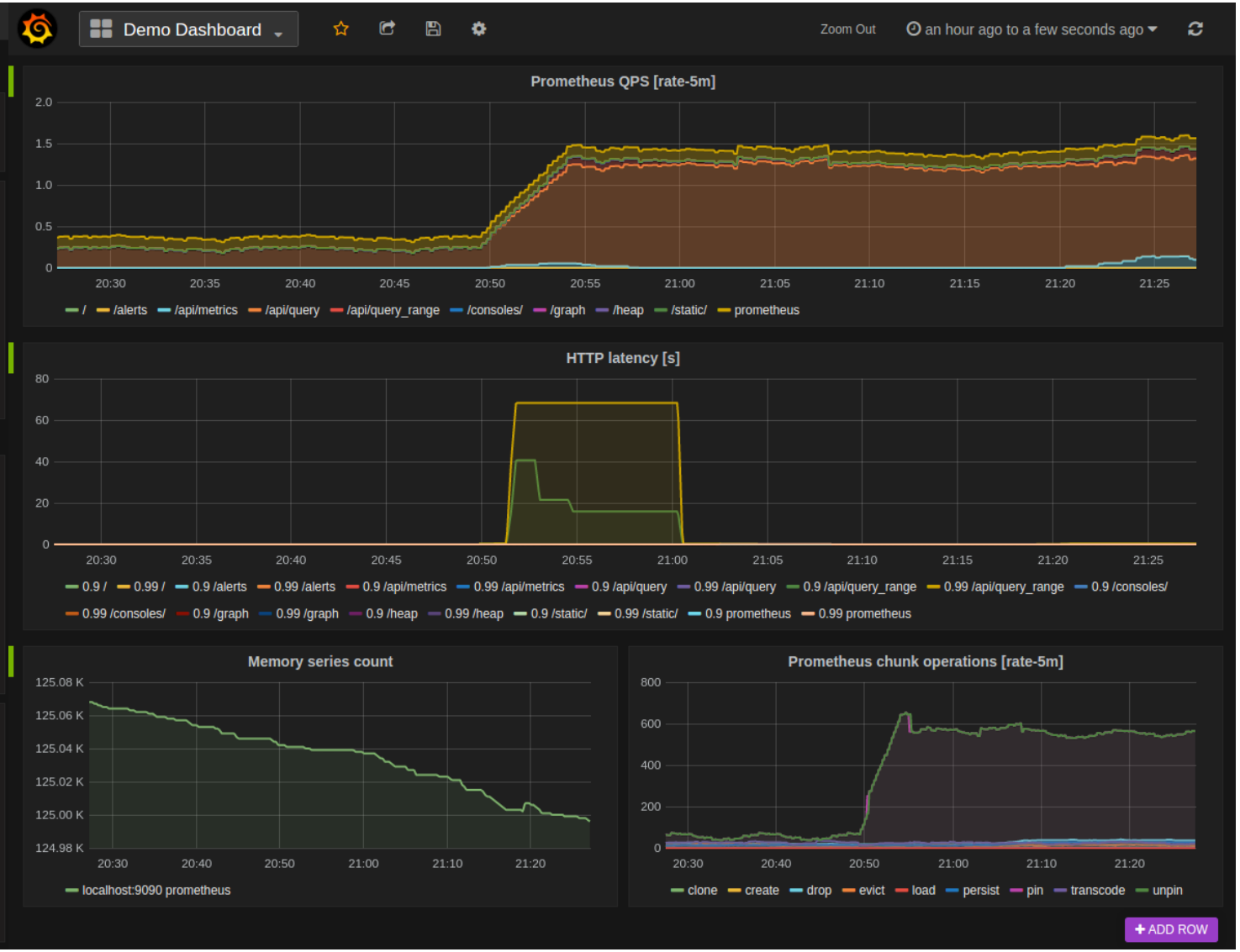
```
// Prometheus client setup
var Prometheus = require('prom-client');
Prometheus.collectDefaultMetrics();

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

```
# HELP nodejs_heap_size_total_bytes Process heap size from Node.js in bytes.
# TYPE nodejs_heap_size_total_bytes gauge
nodejs_heap_size_total_bytes 21217280

# HELP nodejs_heap_size_used_bytes Process heap size used from Node.js in bytes.
# TYPE nodejs_heap_size_used_bytes gauge
nodejs_heap_size_used_bytes 10339824

# HELP nodejs_external_memory_bytes Node.js external memory size in bytes.
# TYPE nodejs_external_memory_bytes gauge
nodejs_external_memory_bytes 147475

# HELP nodejs_heap_space_size_total_bytes Process heap space size total from Node.js in bytes.
# TYPE nodejs_heap_space_size_total_bytes gauge
nodejs_heap_space_size_total_bytes{space="read_only"} 524288
nodejs_heap_space_size_total_bytes{space="new"} 8388608
nodejs_heap_space_size_total_bytes{space="old"} 8097792
nodejs_heap_space_size_total_bytes{space="code"} 1572864
nodejs_heap_space_size_total_bytes{space="map"} 1060864
nodejs_heap_space_size_total_bytes{space="large_object"} 1572864

# HELP nodejs_heap_space_size_used_bytes Process heap space size used from Node.js in bytes.
# TYPE nodejs_heap_space_size_used_bytes gauge
nodejs_heap_space_size_used_bytes{space="read_only"} 35200
nodejs_heap_space_size_used_bytes{space="new"} 1893792
nodejs_heap_space_size_used_bytes{space="old"} 6616496
nodejs_heap_space_size_used_bytes{space="code"} 920288
nodejs_heap_space_size_used_bytes{space="map"} 629464
nodejs_heap_space_size_used_bytes{space="large_object"} 249216

# HELP nodejs_heap_space_size_available_bytes Process heap space size available from Node.js in bytes.
# TYPE nodejs_heap_space_size_available_bytes gauge
nodejs_heap_space_size_available_bytes{space="read_only"} 480384
nodejs_heap_space_size_available_bytes{space="new"} 2230880
nodejs_heap_space_size_available_bytes{space="old"} 1286048
nodejs_heap_space_size_available_bytes{space="code"} 704
nodejs_heap_space_size_available_bytes{space="map"} 80
nodejs_heap_space_size_available_bytes{space="large_object"} 1505500672

# HELP nodejs_version_info Node.js version info.
# TYPE nodejs_version_info gauge
nodejs_version_info{version="v10.19.0",major="10",minor="19",patch="0"} 1

# HELP nodejs_gc_duration_seconds Garbage collection duration by kind, one of major, minor, incremental or weak
# TYPE nodejs_gc_duration_seconds histogram
```

And more...

Prometheus


Host Metrics
Zoom Out
Last 24 hours
Refresh ever...
Host: prometheus.0
Disk Volume: /data
Host Metrics Copy For Demo
Prometheus Stats
sensu-proxy
Load Avg. 1m
0.8
State
up
Memory
15.7 GiB
All disks
1007 GiB
Uptime
10.63 week
CPU usage
guest Current: 0% guest_nice Current: 0% iowait Current: 6.62%
irq Current: 0% nice Current: 0% softirq Current: 0.09%
steal Current: 0.09% system Current: 2.92% user Current: 20.58%
Disk I/O utilization
/dev/xvda /dev/xvda1 /dev/xvdb /dev/xvdb1 /dev/xvdc
Memory usage
used buffers cached free
Network usage
min max avg
eth0 outbound 0 bps 1.18 Mbps 318 kbps
eth0 inbound 0 bps 18.27 Mbps 4.52 Mbps
eth0 drops inbound (right-y) 0 0 0
eth0 drops outbound (right-y) 0 0 0
Filesystem /data usage
min max avg current
used 102 GiB 104 GiB 103 GiB 102 GiB
free 896 GiB 897 GiB 897 GiB 897 GiB

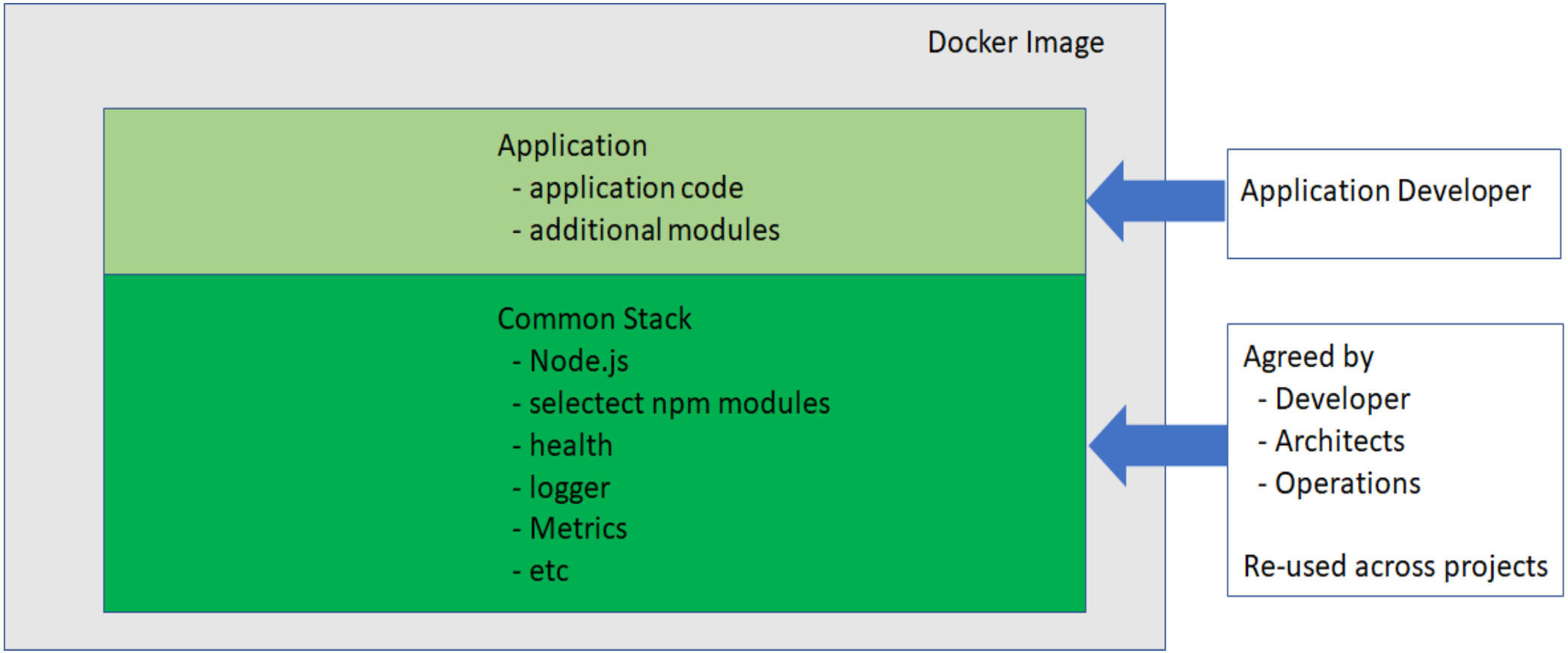

Demo Dashboard
Zoom Out
an hour ago to a few seconds ago
Prometheus QPS [rate-5m]
/ /alerts /api/metrics /api/query /api/query_range /consoles/ /graph /heap /static/ prometheus
HTTP latency [s]
0.9 / 0.99 / 0.9 /alerts 0.99 /alerts 0.9 /api/metrics 0.99 /api/metrics 0.9 /api/query 0.99 /api/query 0.9 /api/query_range 0.99 /api/query_range 0.9 /consoles/
0.99 /consoles/ 0.9 /graph 0.99 /graph 0.9 /heap 0.99 /heap 0.9 /static/ 0.99 /static/ 0.9 prometheus 0.99 prometheus
Memory series count
localhost:9090 prometheus
Prometheus chunk operations [rate-5m]
clone create drop evict load persist pin transcode unpin
+ ADD ROW

Lab Part 1 - Overview

- Build simple express application,
- Add health checking and metrics using CNJS technologies
- Build it into a docker container,
- Deploy it to Kubernetes.
- Connect it to a Prometheus server to monitor it

But do I have to do all that for every Project?

Separation of Concerns…



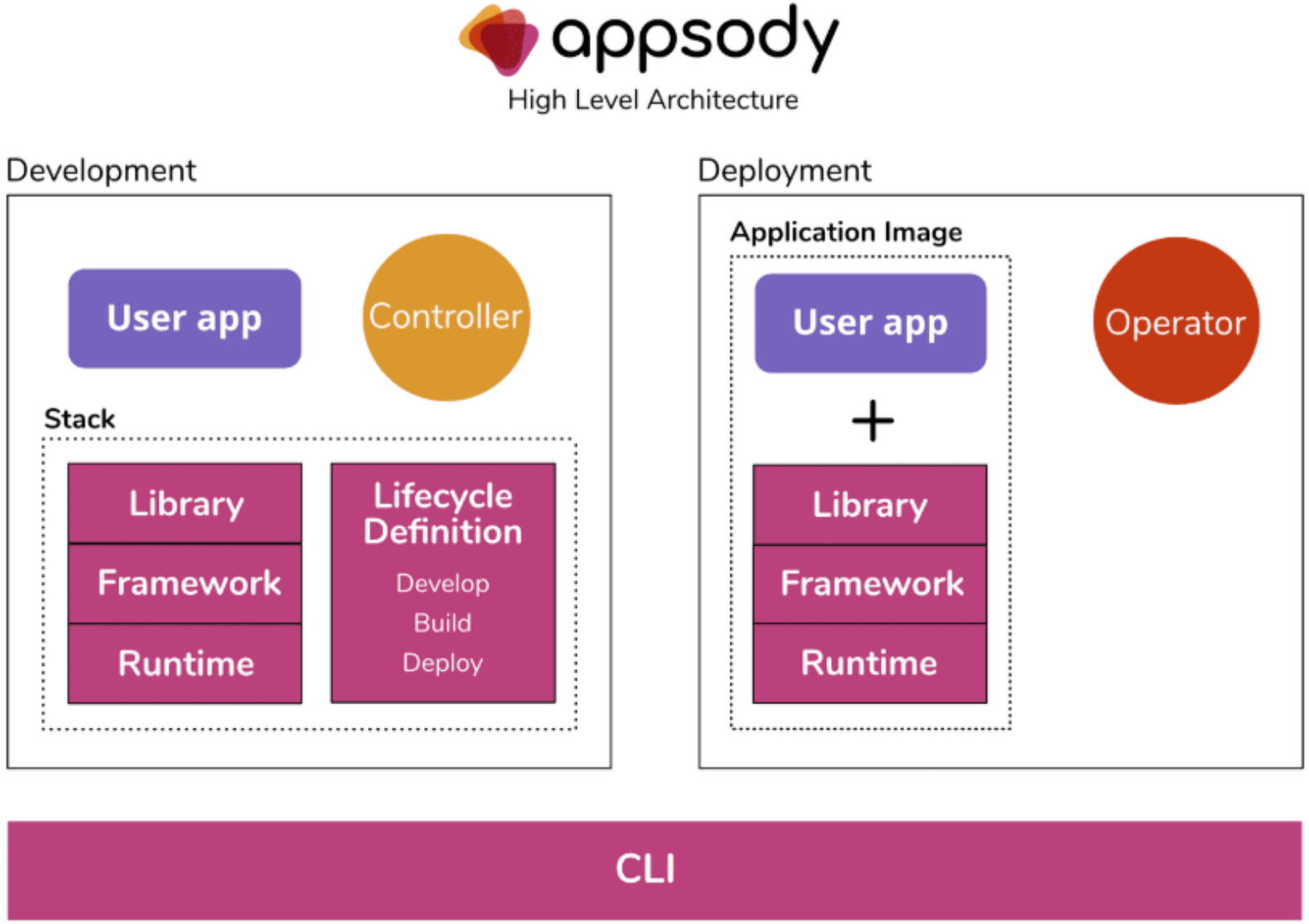
Some Existing Tools

Appsody - https://appsody.dev/

odo - https://developers.redhat.com/products/odo/overview

Tooling to create, develop and deploy
cloud-native applications
using cloud-optimized application stacks.

appsody

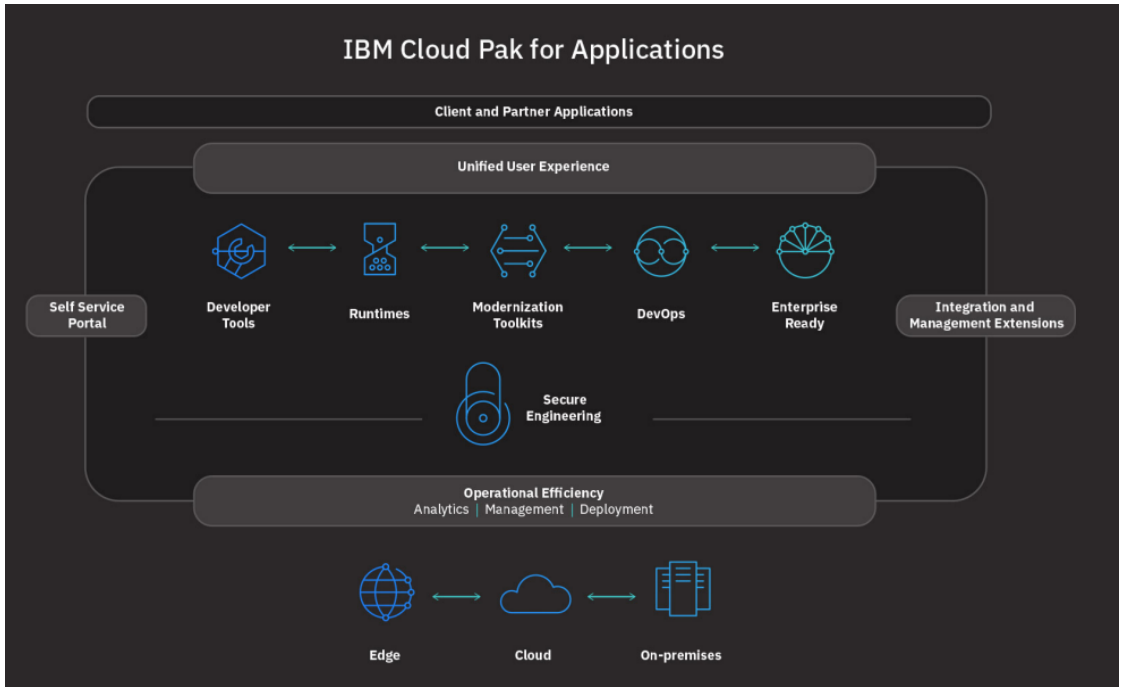

appsody
High Level Architecture
Development
User app
Controller
Stack
Library
Framework
Runtime
Lifecycle Definition
Develop
Build
Deploy
Deployment
Application Image
User app
+
Library
Framework
Runtime
Operator
CLI

Lab Part 2 - Overview

- Install appsody
- Build Express application with appsody nodejs-express stack
- Deploy to Kubernetes
- Check out the build in Cloud Native functionality

IBM Cloud Pak for Applications

- Pulls together components into Enterprise Level Solution



https://www.ibm.com/cloud/cloud-pak-for-applications

- For example
  - OpenShift
  - CodeWind which is a VS Code plug-in https://www.eclipse.org/codewind/
  - Kabanero stacks (UBI based) https://kabanero.io/
  - And much more …..

Lab Part 3 - Overview

- Create Appsody project with Kabanero stack
- Deploy to Kubernetes
- Explore Codewind UI
  - Use Kabanero Stack from Codewind
  - View Application Metrics
  - Run Load Tests using Codewind

On to the Lab…

https://ibm.biz/nodejs-in-the-cloud-think-2020

Copyright and Trademarks