
AWS
re:Invent

CON307-S

Top 5 container and Kubernetes best practices

Grace Andrews
Solutions Consultant
New Relic

Safe Harbor

This presentation and the information herein (including any information that may be incorporated by reference) is provided for informational purposes only and should not be construed as an offer, commitment, promise or obligation on behalf of New Relic, Inc. ("New Relic") to sell securities or deliver any product, material, code, functionality, or other feature. Any information provided hereby is proprietary to New Relic and may not be replicated or disclosed without New Relic's express written permission.

Such information may contain forward-looking statements within the meaning of federal securities laws. Any statement that is not a historical fact or refers to expectations, projections, future plans, objectives, estimates, goals, or other characterizations of future events is a forward-looking statement. These forward-looking statements can often be identified as such because the context of the statement will include words such as "believes," "anticipates," "expects" or words of similar import.

Actual results may differ materially from those expressed in these forward-looking statements, which speak only as of the date hereof, and are subject to change at any time without notice. Existing and prospective investors, customers and other third parties transacting business with New Relic are cautioned not to place undue reliance on this forward-looking information. The achievement or success of the matters covered by such forward-looking statements are based on New Relic's current assumptions, expectations, and beliefs and are subject to substantial risks, uncertainties, assumptions, and changes in circumstances that may cause the actual results, performance, or achievements to differ materially from those expressed or implied in any forward-looking statement. Further information on factors that could affect such forward-looking statements is included in the filings New Relic makes with the SEC from time to time. Copies of these documents may be obtained by visiting New Relic's Investor Relations website at ir.newrelic.com or the SEC's website at www.sec.gov.

New Relic assumes no obligation and does not intend to update these forward-looking statements, except as required by law. New Relic makes no warranties, expressed or implied, in this presentation or otherwise, with respect to the information provided.

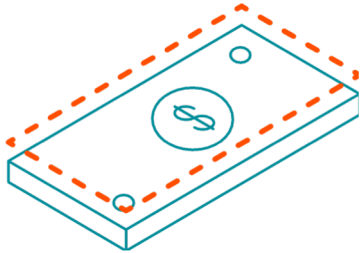
Grace Andrews

Solutions Consultant

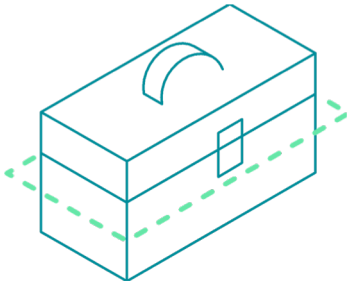
Living in Portland, OR

Passionate about

Kubernetes helps you
move faster



Kubernetes is
cost-efficient



Kubernetes is
portable

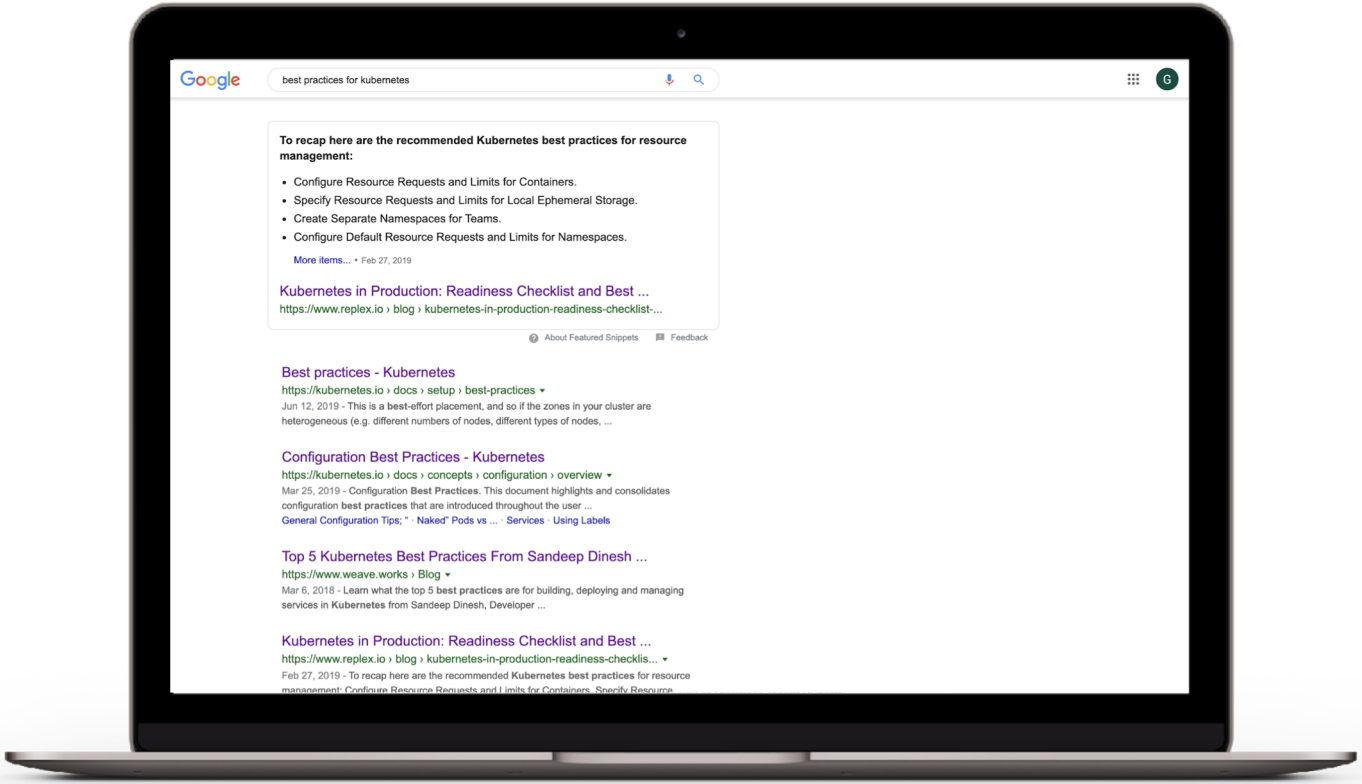
Kubernetes adoption journey

"Help, I need to Kubernetes"

Stable Kubernetes Cluster

Kubernetes Promised Land

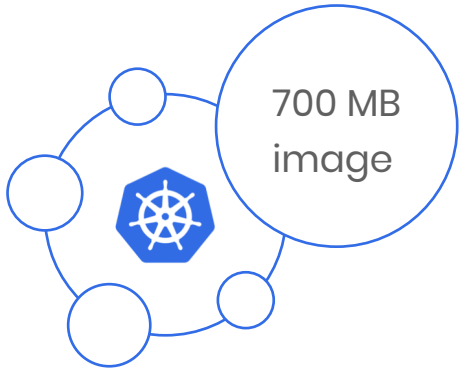
In the beginning… you know the rest



Best practices

- Build small container images
- Leverage namespaces
- Set up health checks
- Use resource requests & limits; multi-functional alerts
- Understand the context of problems

Build small container images



Why?

- Improve time-to-build & time-to-pull
- Fewer security vulnerabilities

Build small container images

1 **Kubernetes builds**

Kubernetes cluster explorer

Why?

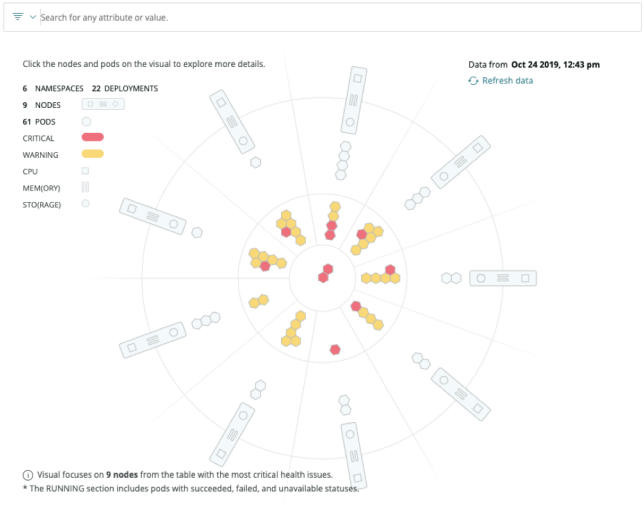
- Quicker builds
- Smaller attack surface
- Less storage
- Faster image pulls/cold starts

How?

- Identify your app(s) need
- Choose the appropriate base image
- Avoid unnecessary resources (packages + files)

Base node.js image	Alpine
FROM node:latest WORKDIR /app COPY . . RUN npm install -- prod EXPOSE 8080 CMD npm start	FROM node:alpine WORKDIR /app COPY . . RUN npm install -- prod EXPOSE 8080 CMD npm start
700 MB	**65 MB**

Reduce time in PENDING state



Leverage the Docker build cache

Layer 5 | *CMD npm start*

Layer 4 | *Expose 8080*

Layer 3 | *RUN npm install -- prod*

Layer 2 | *COPY. .*

Layer 1 | *WORKDIR/app*

Base image | *FROM node:alpine*

Layer 6 | *CMD npm start*

Layer 5 | *Expose 8080*

Layer 4 | *COPY SRC_CODE SRC_CODE*

Layer 3 | *RUN npm install --prod*

Layer 2 | *COPY package.json .*

Layer 1 | *WORKDIR /app*

Base image | *FROM node:alpine*

Add your source code as late as possible

Base image and dependencies can be cached

Using multi-stage builds

Layer X | *# Build your final image*

Layer 1 | *COPY –from=dependencies /app/prod_node_modules ./node_modules*

Layer 0 | *FROM base AS release*

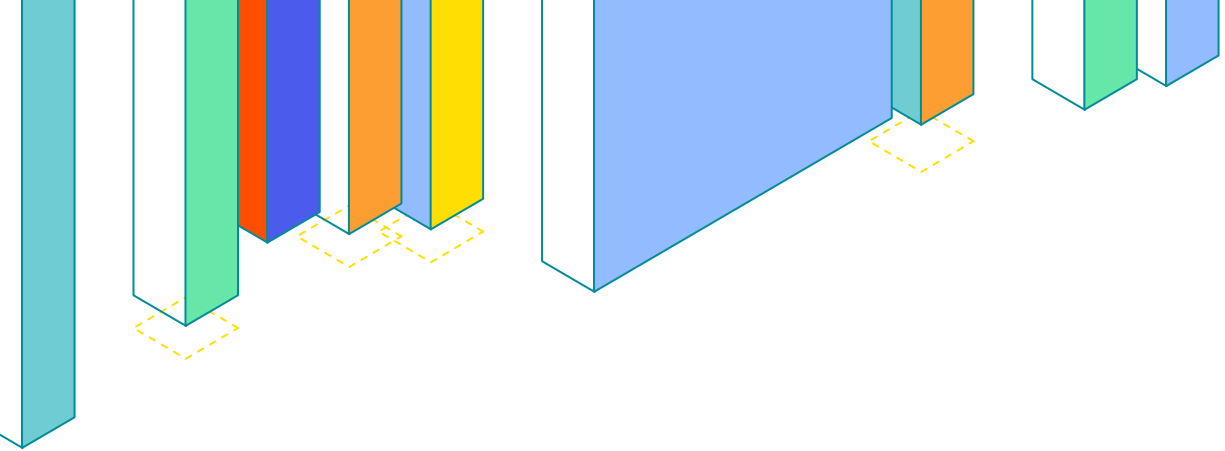
Layer X | *# Build your dependencies*

Layer 0 | *FROM base AS dependencies*

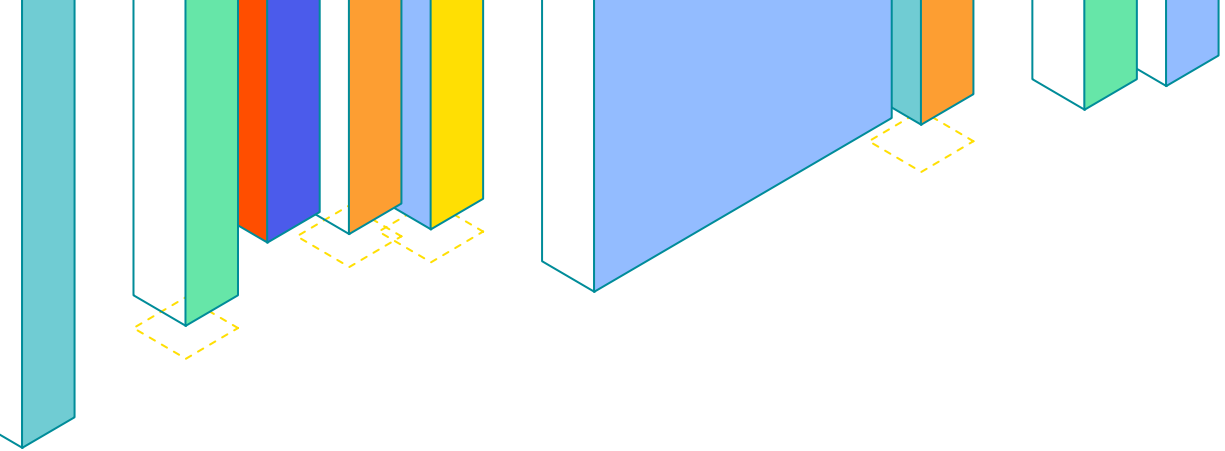
Layer x | *# Build the base image*

Layer 0 | *FROM node:alpine AS base*

Multi-stage builds! Break down your Dockerfile in multiple stages
The final image will be based on the latest FROM



Back to the cluster



Example

What's in a name?

Small team

PROD

DEV

Multiple teams

SERVICE A PROD

SERVICE A DEV

SERVICE B PROD

SERVICE B DEV

Enterprise

SERVICE A PROD

SERVICE A DEV

SERVICE B PROD

SERVICE B DEV

SERVICE C PROD

SERVICE C DEV

SERVICE D PROD

Kubernetes observability – Take advantage of namespaces

2 **Kubernetes namespace**

Error profiles, system health

Why?

Improve manageability & security

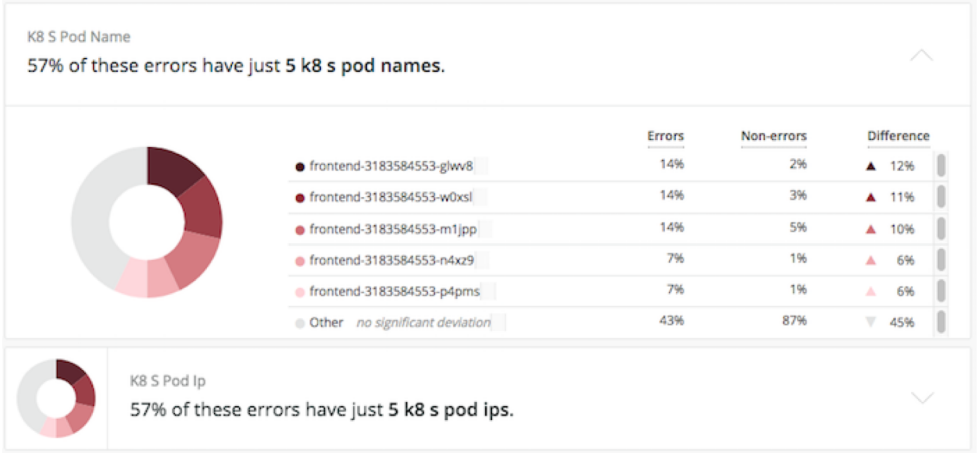
- By isolating Kubernetes resource for different teams

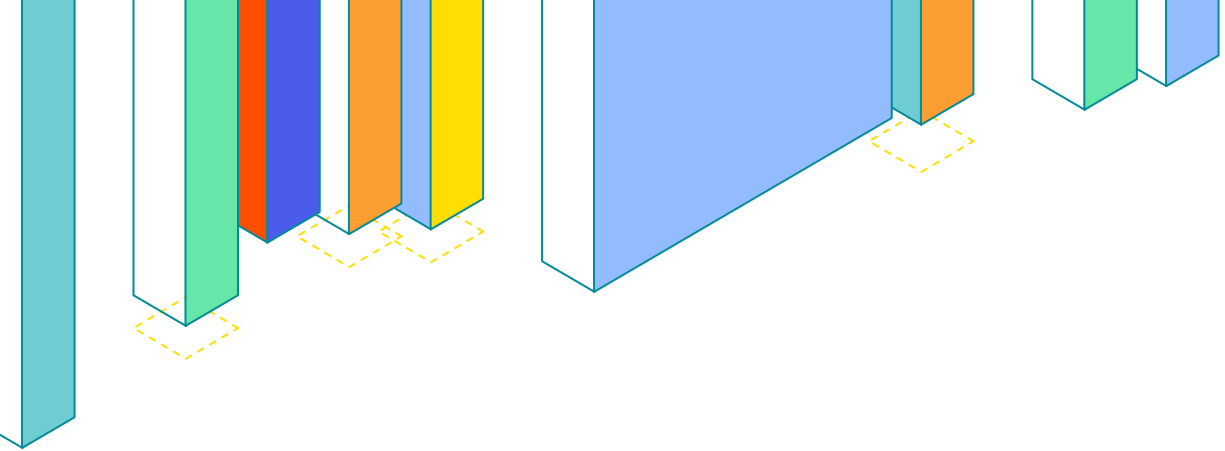
Improve performance

- Kubernetes API working on smaller set of objects

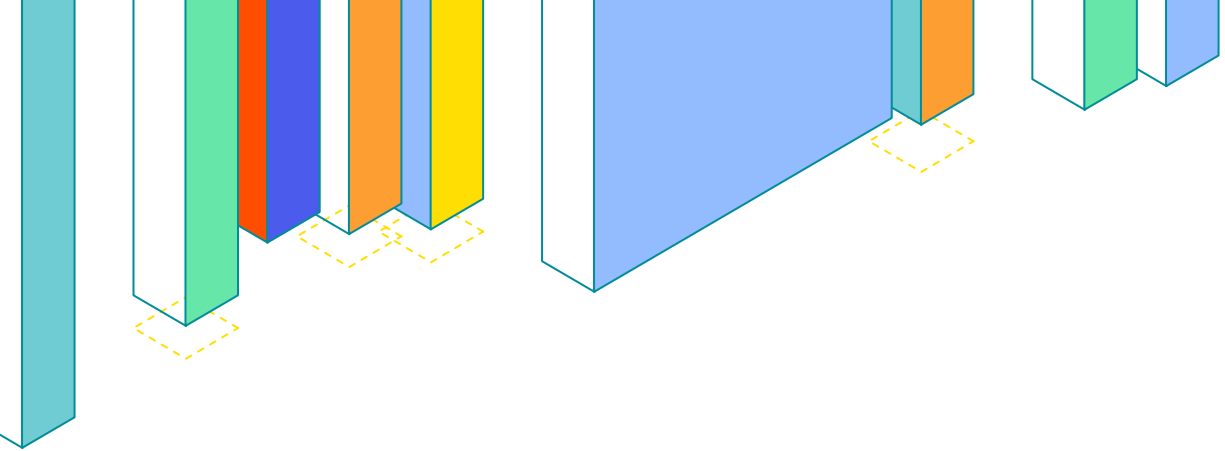
How?

- Modify config file (YAML)
- Update YOUR_CLUSTER_NAME
- Kubectl apply to initiate the changes

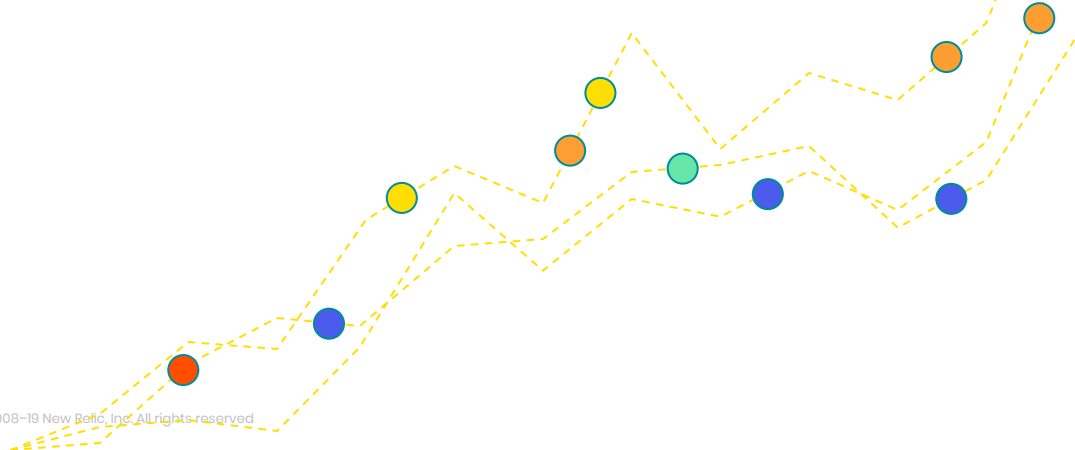




Back to the cluster



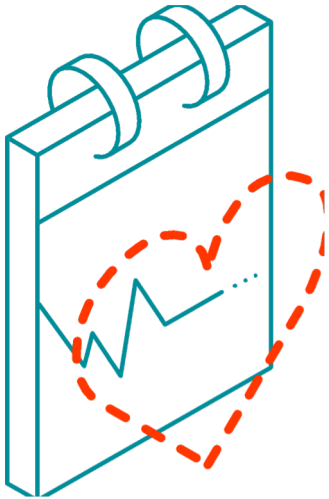
Example



Set up health checks

Readiness probes

Ready, set, GO!



Liveness probes

Dead or alive?

Kubernetes observability – Health checks

3 **Kubernetes health checks**

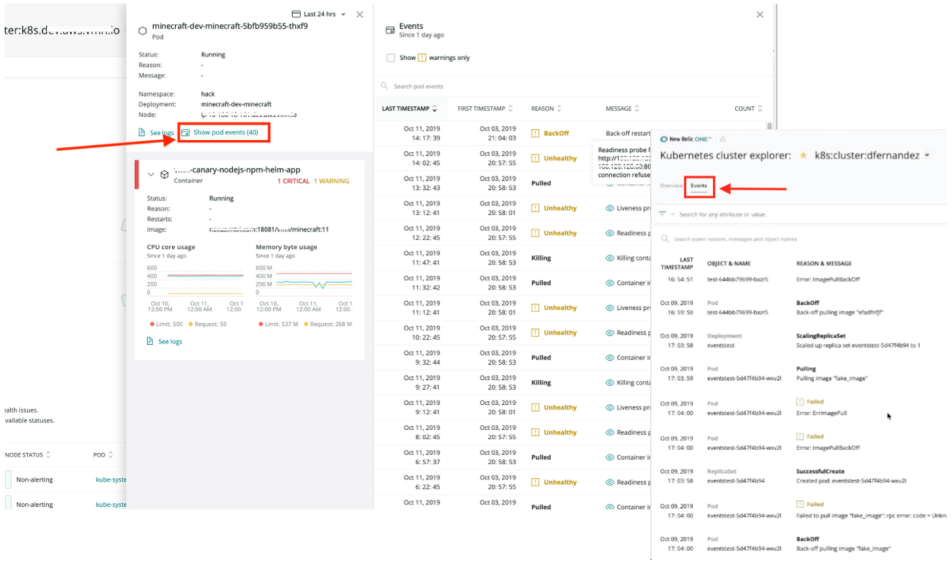
Track vital cluster events

Why?

- Faster troubleshooting and issue resolution
- Watch all events happening in your cluster
- Track specific events such as scaling events

How?

- Download YAML file
- Modify the YAML file with the proper probes
- Remember to set initialDelaySeconds
- Use kubectl apply to update changes



Health checks

```
readinessProbe:
  exec:
    command:
    - cat
    - /tmp/healthy
  initialDelaySeconds: 5
  periodSeconds: 5
```

Readiness probes

Is the application ready to accept traffic or not?

Tip: A synthetics check can travel the full path from the load balancer through the traffic proxy to the container and verify that the network path is working appropriately.

Health checks

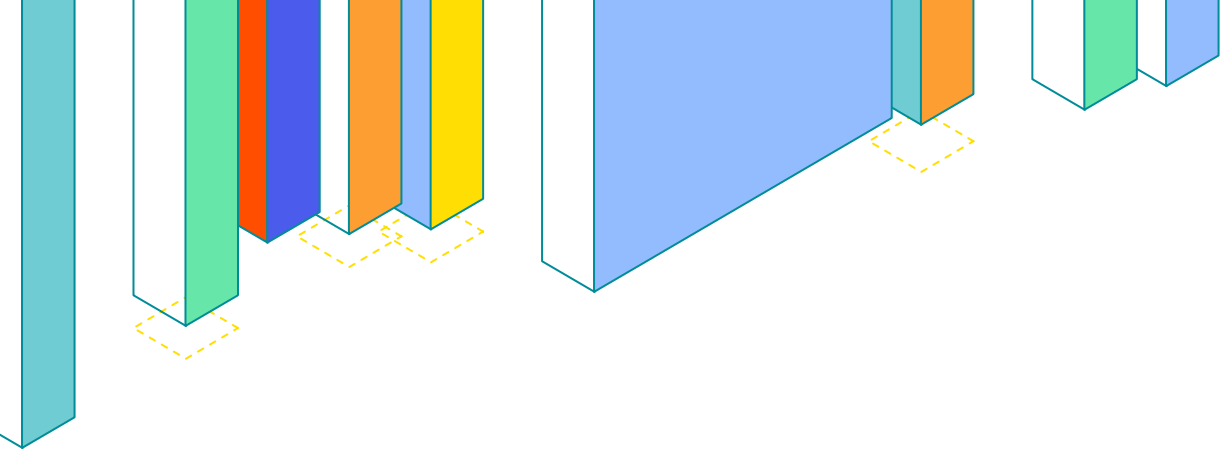
Liveness probes
Is our application even alive, or is it dead?

Why is our application dead?
Correlate your k8s data with application data!

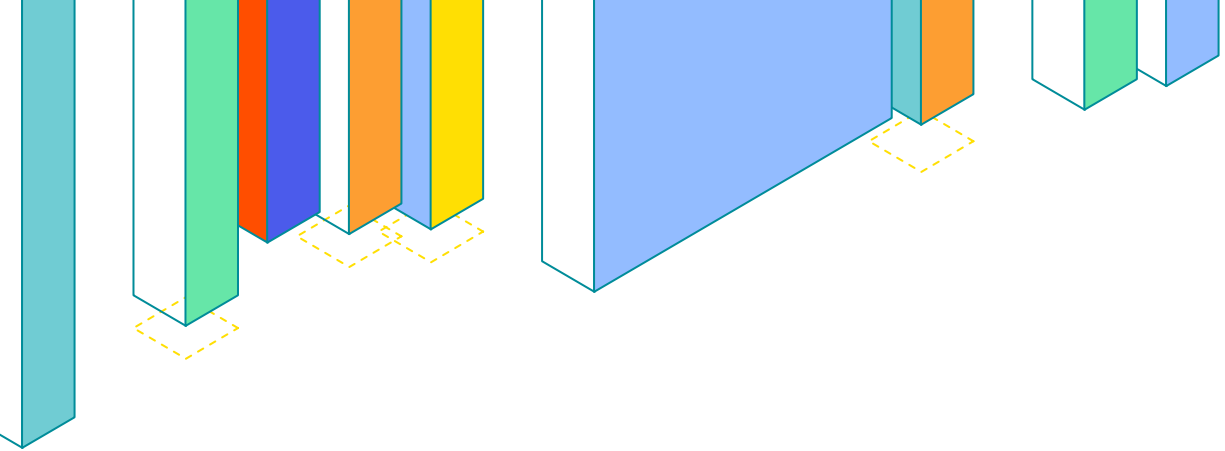
pods/probe/exec-liveness.yaml

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    test: liveness
  name: liveness-exec
spec:
  containers:
  - name: liveness
    image: k8s.gcr.io/busybox
    args:
    - /bin/sh
    - -c
    - touch /tmp/healthy; sleep 30; rm -rf /tmp/healthy; sleep 600
    livenessProbe:
      exec:
        command:
        - cat
        - /tmp/healthy
      initialDelaySeconds: 5
      periodSeconds: 5
```

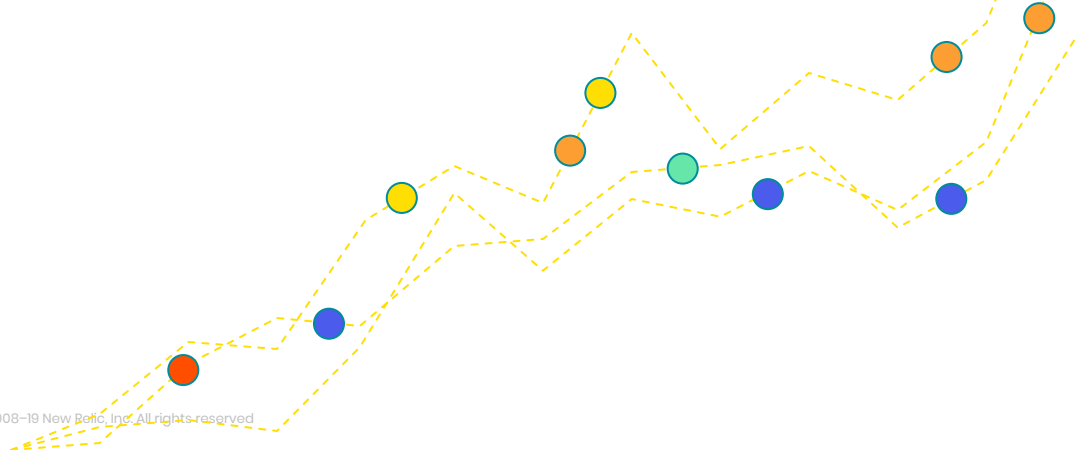
Remark:
Set **initialDelaySeconds** for liveness probe or pod will be restarted before it is ready



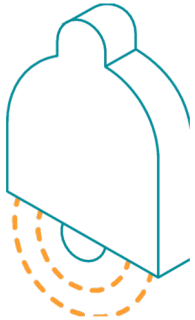
Back to the cluster



Example



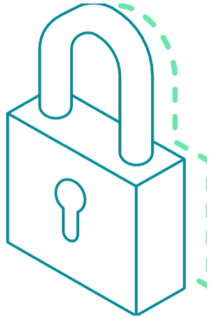
Set up requests & limits



Requests

What the container
is guaranteed to get

Example: When a container requests a resource, Kubernetes will only schedule it on a node that can give it that resource



Limits

Make sure a container never goes above a certain value

The container can only go up to the limit, and then it's restricted

Kubernetes observability – Requests & limits

4 **Kubernetes resource requests & limits**

Warning + alerts

Why?

Lack of resources limits creates pod instability and possible termination

- Pod over CPU limit: Throttled
- Pod over memory limit: Terminated
- Pod in PENDING state if resources are not available

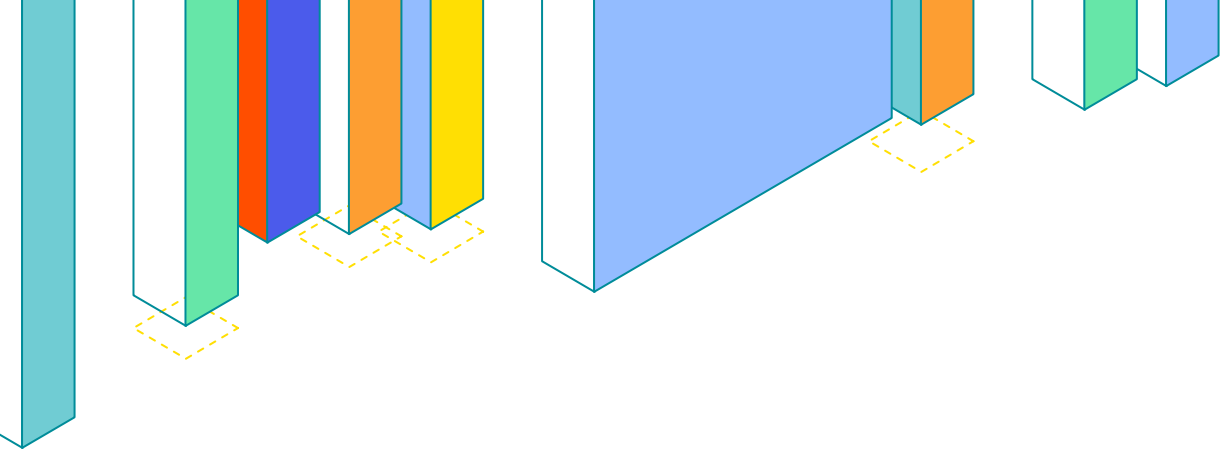
Inappropriately sized requests can cause scheduling issues

- If a value larger than the core count of your biggest node is set, your pod will never be scheduled

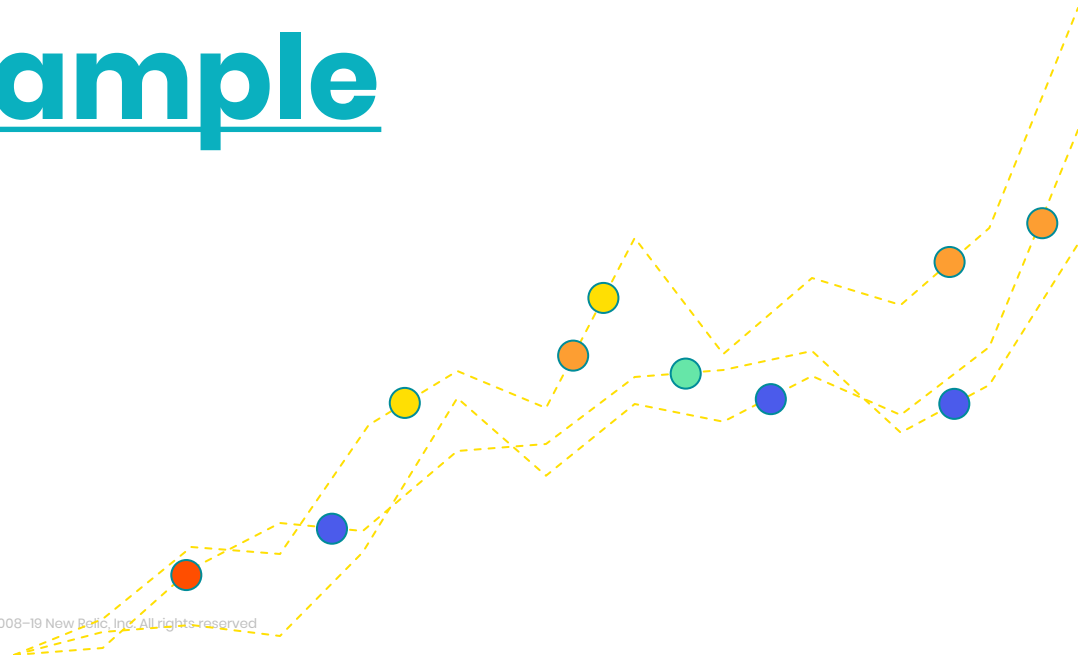
How?

1. Specify limits and requests within your config file (Kubernetes.io)
2. Use kubectl apply to push changes
3. Use kubectl describe nodes to see a list of requests and limits

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  containers:
  - name: db
    image: mysql
    env:
    - name: MYSQL_ROOT_PASSWORD
      value: "password"
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
  - name: wp
    image: wordpress
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```



Example

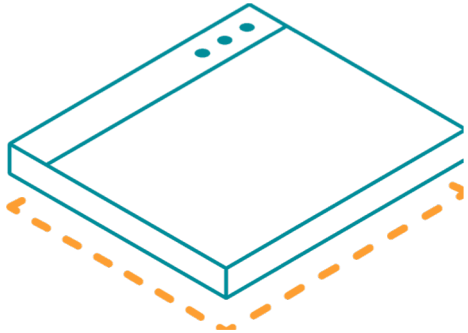


Leveraging logs

Gain visibility
What's happening in the cluster?

Gain visibility
Are the pods behaving properly?

Gain stability
Has my host been compromised?



Kubernetes observability – Understanding problems

5 Kubernetes logging

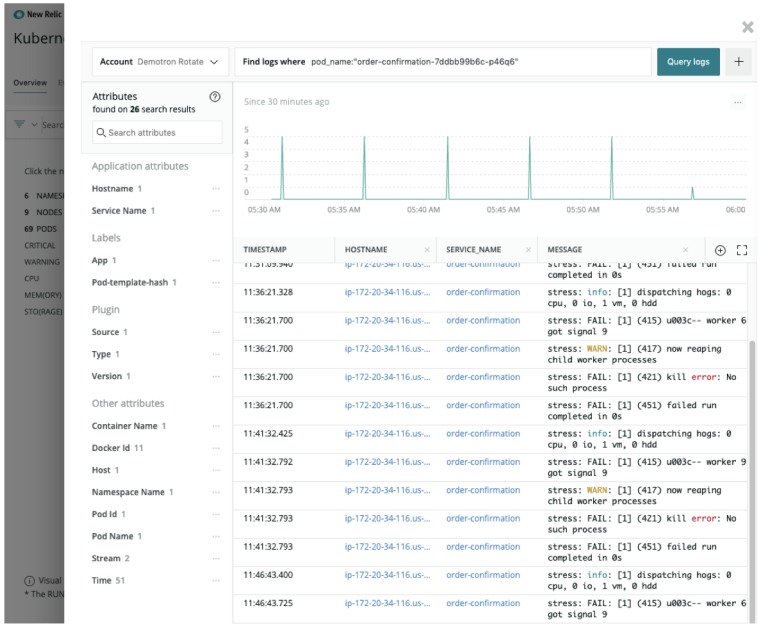
No more context switching
Logs in context
Fast(!) log search

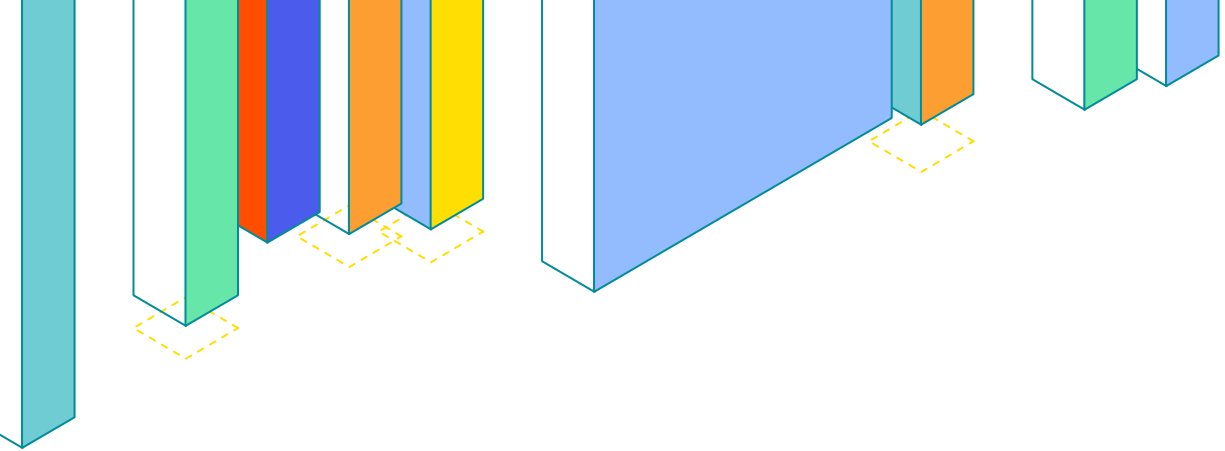
Why?

- Provide more context for environmental behaviors
- Faster troubleshooting when problems arise
- Can be used at the node, pod, and cluster levels

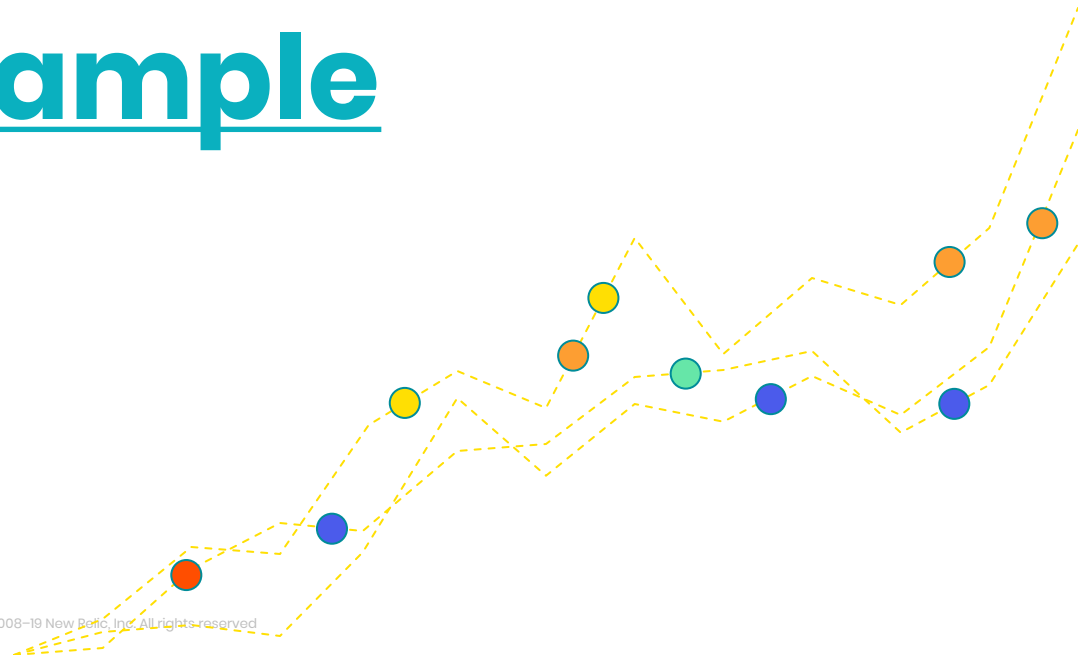
How?

1. Install log output plugin for Kubernetes
2. Update with proper licensing information
3. Use kubectl apply to begin ingest



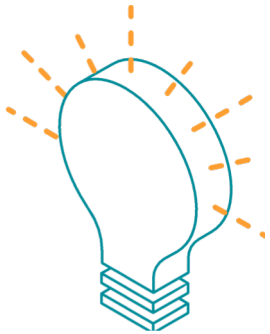


Example



What we learned

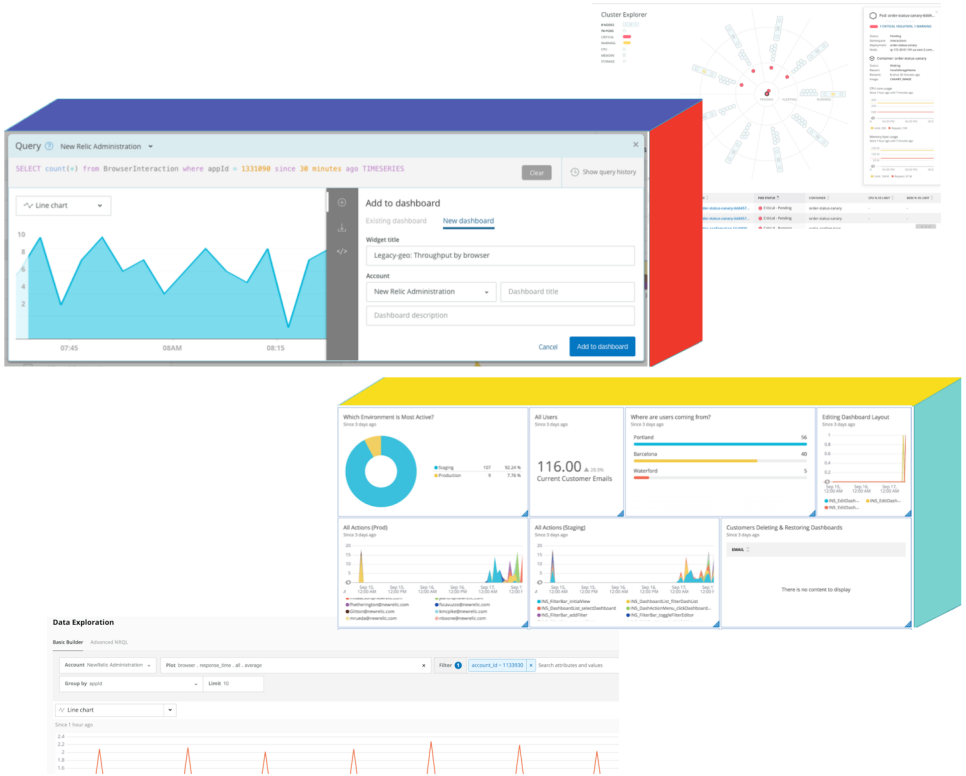
- Think + build small
- Names matter
- Health is wealth
- Ring the alarm: Requests & limits
- Context is gold



New Relic®

Thank You

Grace Andrews
gandrews@newrelic.com



Please complete the session survey in the mobile app.