Kube Me This!

Kubernetes best practices

Karthik Gaekwad

@iteration1

Head of Cloud Native Engineering

Hello

- I’m Karthik Gaekwad
- Head of Cloud Native Engineering

VERICA

@iteration1

Hello

I'm Karthik Gaekwad

Oracle Cloud: Developer on the Managed Kubernetes Team + Developer Relations

Author of devops and Kubernetes courses on LinkedIn Learning

Super popular helloworld docker container

@iteration1

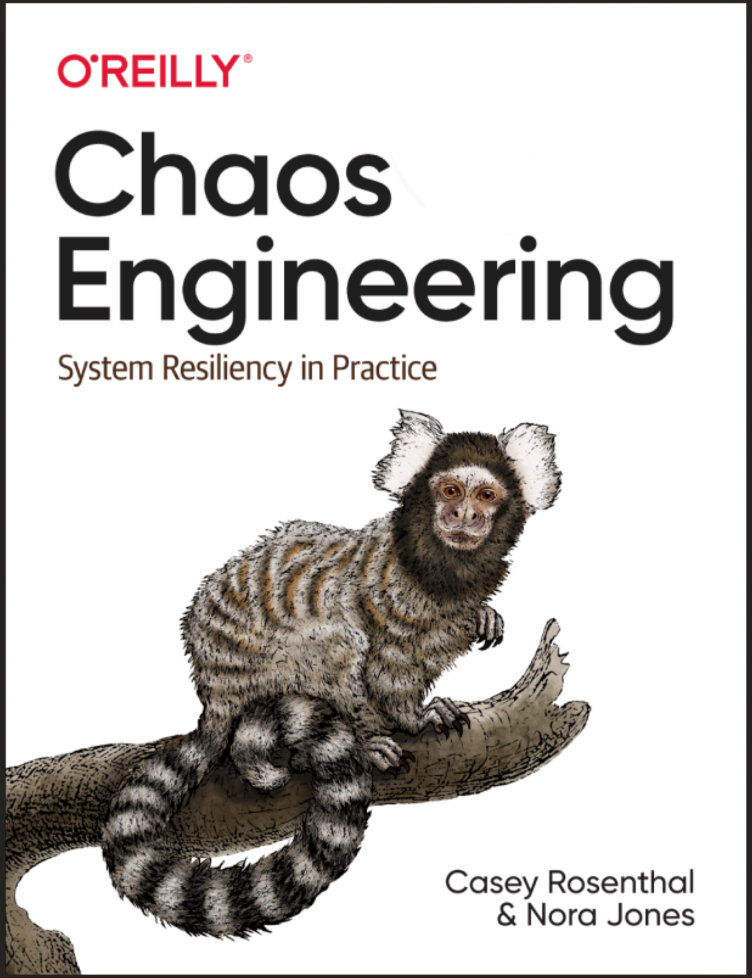
Chaos Engineering

https://verica.io/book/

Want to learn more?

Get the book in 5 minutes!

verica.io/book


O'REILLY

Chaos Engineering

System Resiliency in Practice

Casey Rosenthal & Nora Jones

Today we will

We’re going to talk about Kubernetes…

Break this down into 3 pillars…

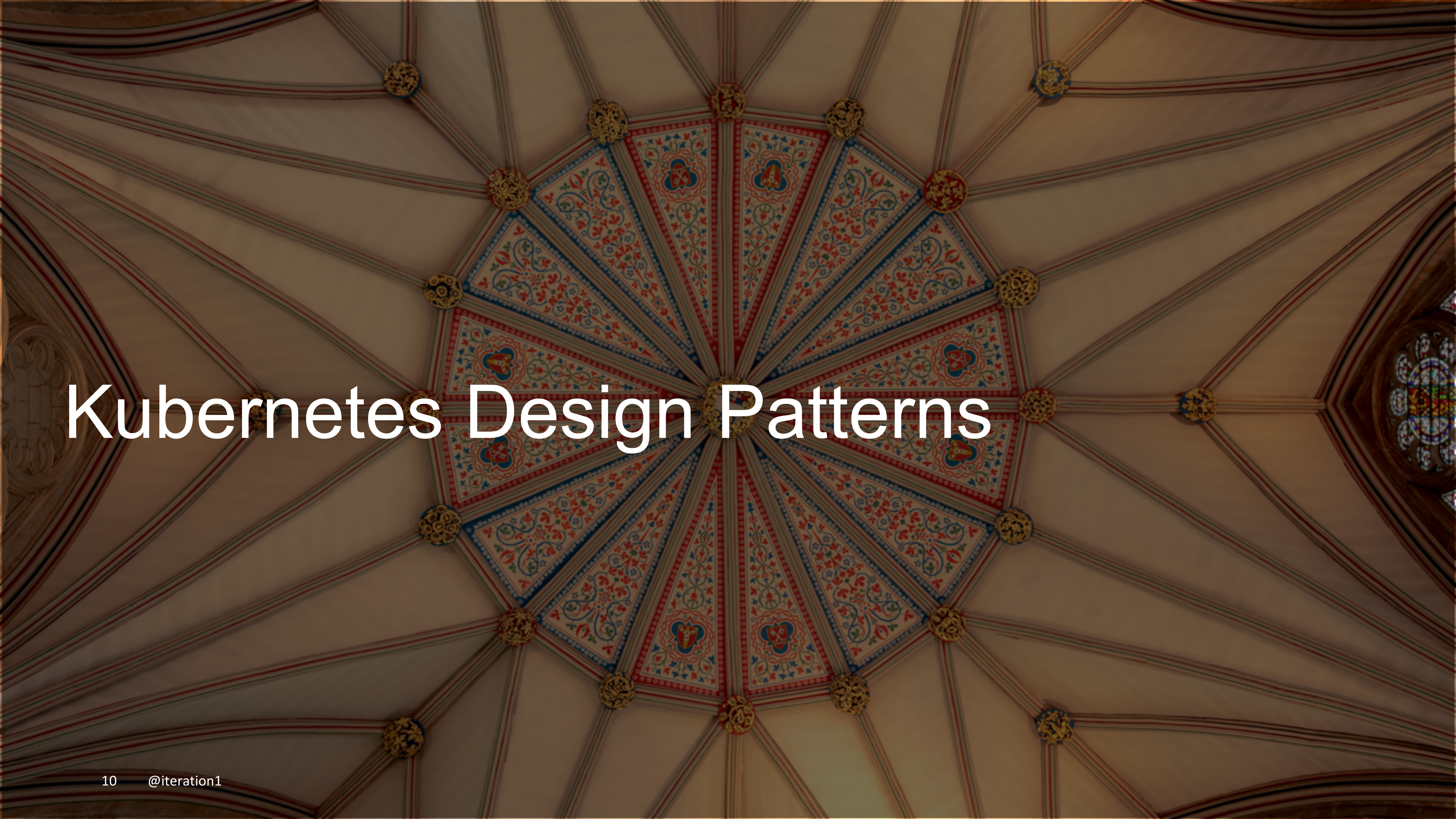
- Development and Architecture
- Devops
- Enterprise Transformation

Development & Architecture

MICROSERVICES
HD
HISTORY.COM
imgflip.com

Microservices Design

- Start with Twelve-Factor App design
- https://12factor.net
- Based on the principals of software design and deployment at Heroku
- Development best practice that synergizes with devops engineers



Kubernetes Design Patterns

Kubernetes Deployments

What??

- Most common K8s object that is used for applications running in Kubernetes.
- Deployment is a defined specification that is used to create replica sets and associated pods.

Kubernetes Deployments

“I’m converting an application (monolith) to a Kubernetes based architecture, what should my deployment look like?”

Kubernetes Deployments

"I'm converting an application (monolith) to a Kubernetes based architecture, what should my deployment look like?"

2 Choices:

Single deployment model

Multi deployment model

Kubernetes Deployments

Single

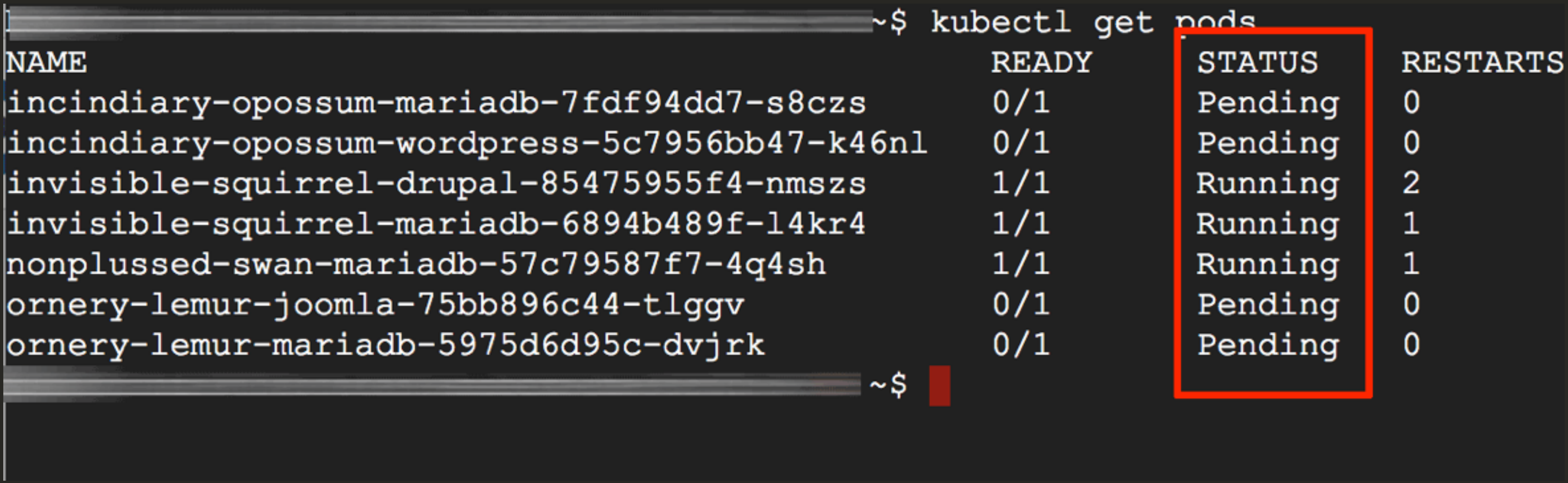
- One single object for your whole application, backed by multiple pods behind the scenes.
- Think of this like one Java WAR file for all parts of your application.

Multi

- Multiple independent deployments for a larger application.
- Components must work together (microservices based architecture)
- This is like having many WAR files for your app.

Working with multiple deployments

Sometimes this can get hairy….



```
[redacted]~$ kubectl get pods
NAME                                         READY     STATUS    RESTARTS
incindiary-opossum-mariadb-7fdf94dd7-s8czs   0/1       Pending   0
incindiary-opossum-wordpress-5c7956bb47-k46nl 0/1      Pending   0
invisible-squirrel-drupal-85475955f4-nmszs   1/1       Running   2
invisible-squirrel-mariadb-6894b489f-l4kr4   1/1       Running   1
nonplussed-swan-mariadb-57c79587f7-4q4sh     1/1       Running   1
ornery-lemur-joomla-75bb896c44-tlggv         0/1       Pending   0
ornery-lemur-mariadb-5975d6d95c-dvjrk        0/1       Pending   0
[redacted]~$
```

Working with multiple deployments

- Consider using liveness and readiness probes.
- Readiness probe
  - User defined health check that tells Kubernetes when the container is ready to serve request.
  - K8s will route traffic to it once it's "ready"
- Liveness probe:
  - User defined health check to indicate whether a container is running.
  - If probe fails, K8s will kill the container and spawn a new one based on the restart policy
- Read more: https://kubernetes.io/docs/concepts/workloads/pods/pod-lifecycle/#container-probes

Working with multiple deployments

- Consider using version endpoints for your pods/containers.
- Can be any defined version strategy= git hash or a user defined version.
- Assists with identification of what is actually running, and reference back to the source code.
- Allows easier debugging, especially when there are multiple teams working on a single deployment, with frequent independent releases.

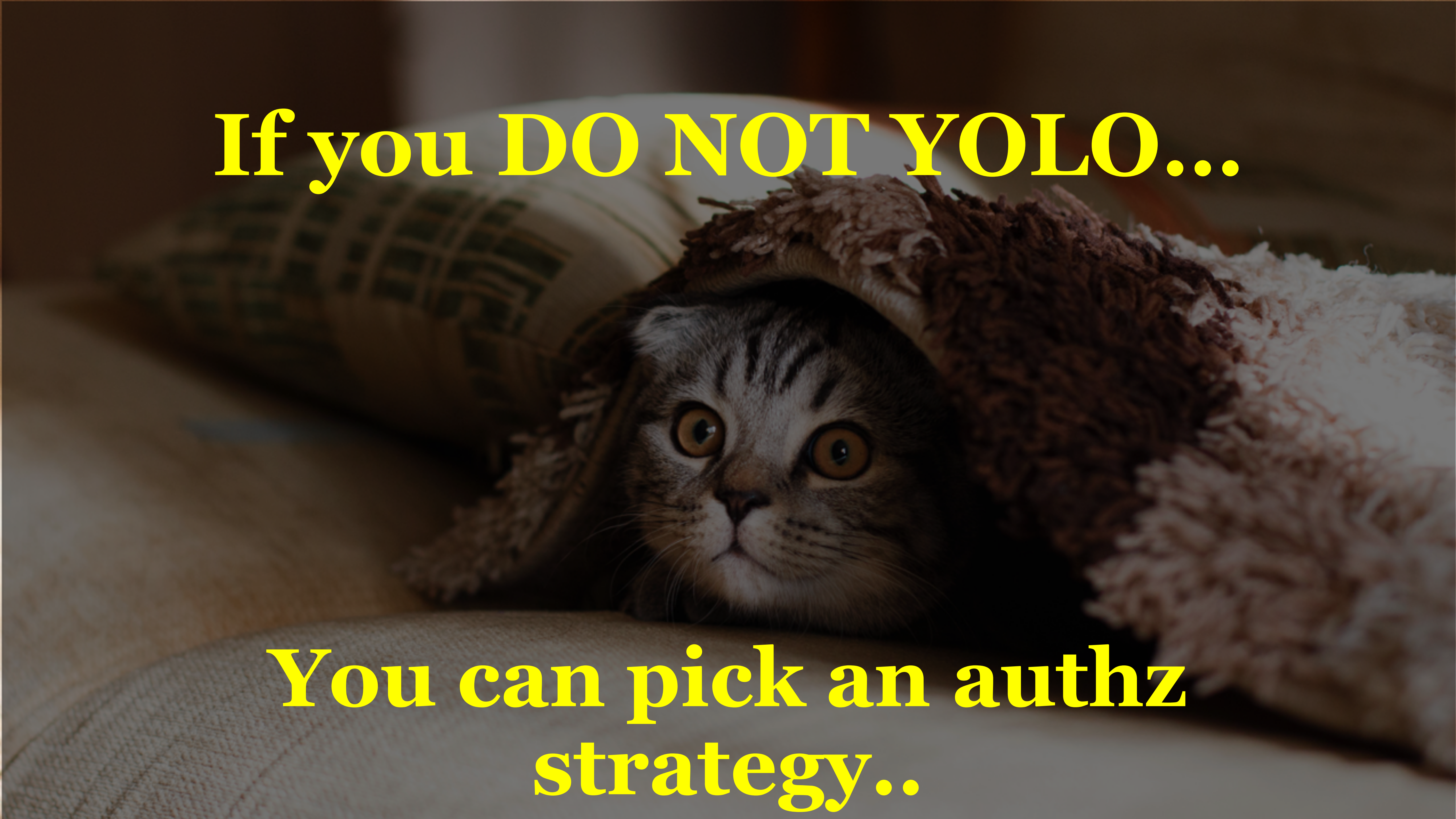
Authentication and Authorization

Authentication and Authorization

- Problem: The concept of a user doesn’t exist in K8s. You have to self manage..
- Do you know how you are authenticating with Kubernetes?
- Many ways to Authenticate
  - Client Certs
  - Static token file
  - Service Account tokens
  - OpenID
  - Webhook Mode
  - And more (https://kubernetes.io/docs/reference/access-authn-authz/authentication/)

Goal: Pick a strategy that fits your use case

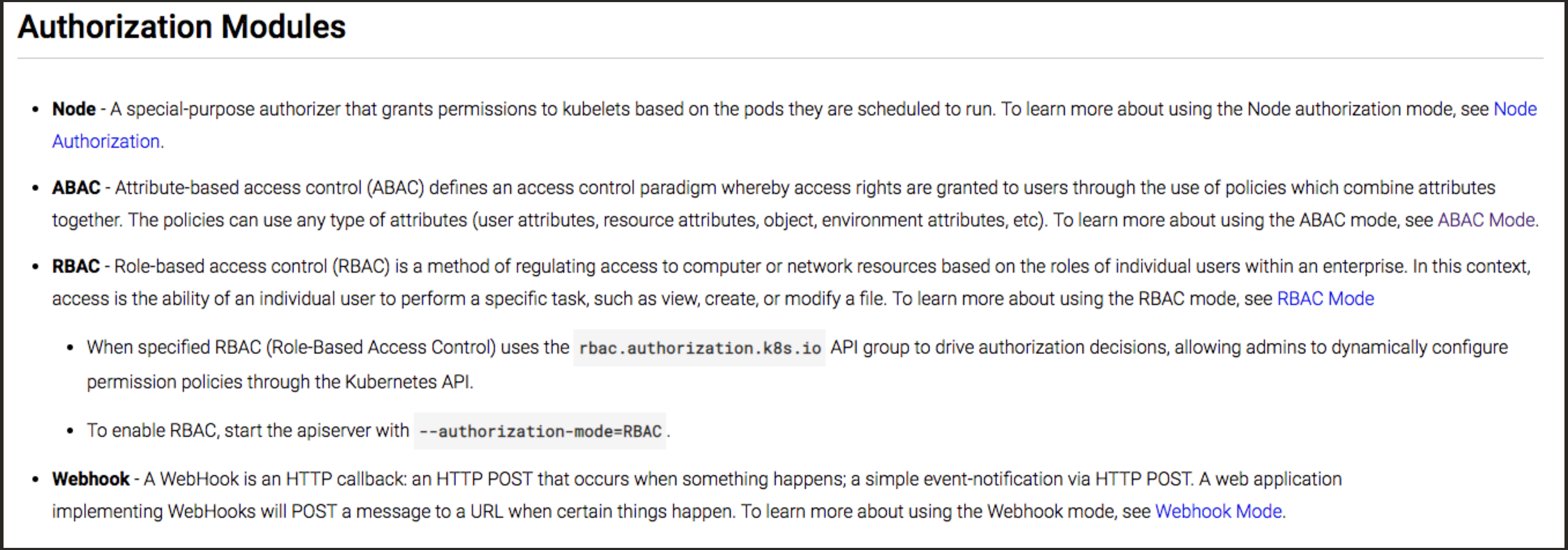
Whatever you do, DO NOT YOLO!



If you DO NOT YOLO…

You can pick an authz strategy..

Authentication and Authorization



Authorization Modules

- **Node** - A special-purpose authorizer that grants permissions to kubelets based on the pods they are scheduled to run. To learn more about using the Node authorization mode, see Node Authorization.
- **ABAC** - Attribute-based access control (ABAC) defines an access control paradigm whereby access rights are granted to users through the use of policies which combine attributes together. The policies can use any type of attributes (user attributes, resource attributes, object, environment attributes, etc). To learn more about using the ABAC mode, see ABAC Mode.
- **RBAC** - Role-based access control (RBAC) is a method of regulating access to computer or network resources based on the roles of individual users within an enterprise. In this context, access is the ability of an individual user to perform a specific task, such as view, create, or modify a file. To learn more about using the RBAC mode, see RBAC Mode
  - When specified RBAC (Role-Based Access Control) uses the `rbac.authorization.k8s.io` API group to drive authorization decisions, allowing admins to dynamically configure permission policies through the Kubernetes API.
  - To enable RBAC, start the apiserver with `--authorization-mode=RBAC`.
- **Webhook** - A WebHook is an HTTP callback: an HTTP POST that occurs when something happens; a simple event-notification via HTTP POST. A web application implementing WebHooks will POST a message to a URL when certain things happen. To learn more about using the Webhook mode, see Webhook Mode.

https://kubernetes.io/docs/reference/access-authn-authz/authorization/

Authentication and Authorization

- Pro tip: Nobody uses ABAC anymore. Don’t be that guy….

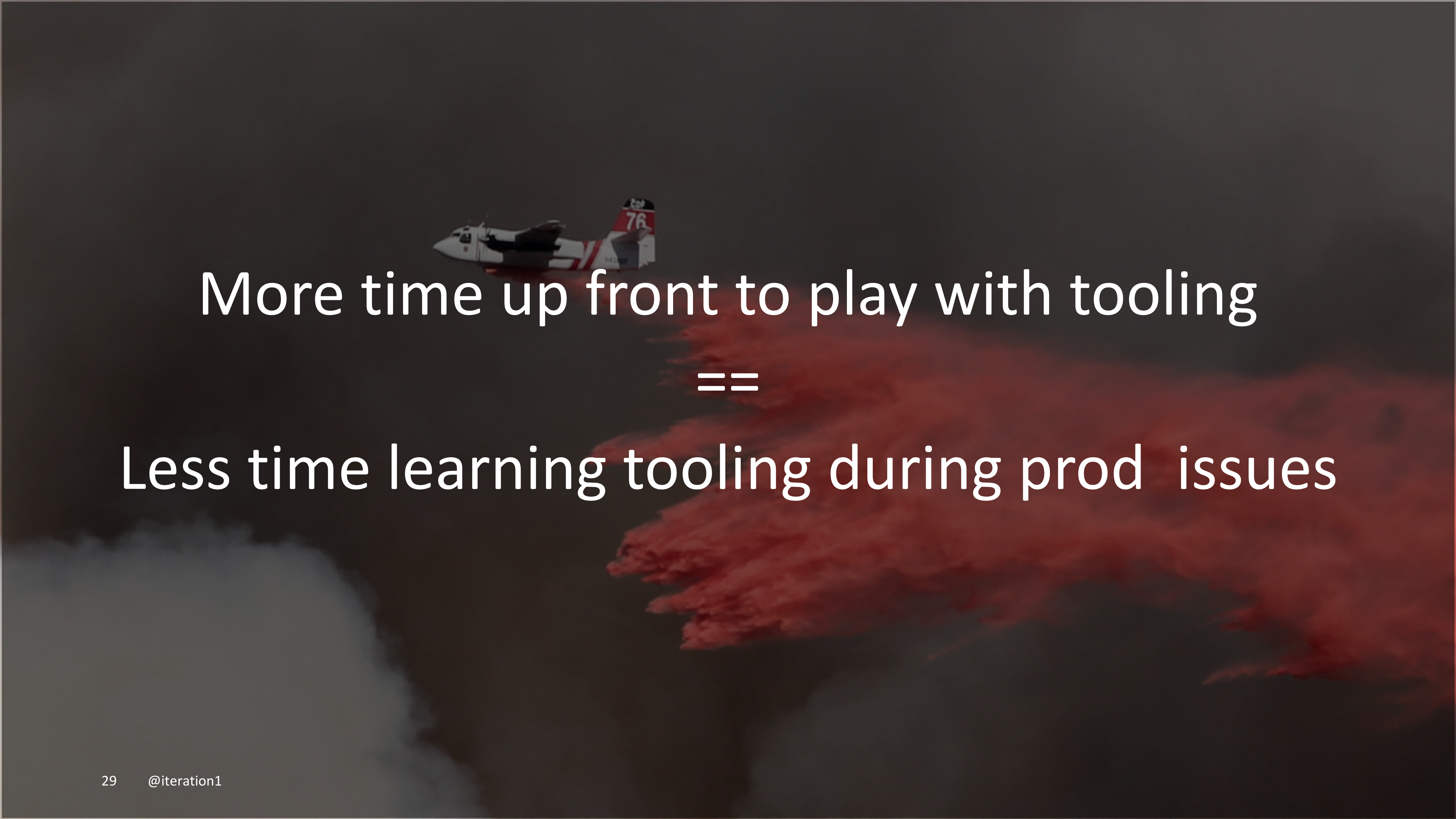
- RBAC is the defacto standard
  - Based on roles and role bindings
  - Good set of defaults: https://github.com/uruddarraju/kubernetes-rbac-policies

- Can use multiple authorizers together, but can get confusing.
  - 1st authorizer to authorize passes authz

Logging and Monitoring

Logging and Monitoring

- kubectl logs and tail commands only takes you so far…
- Invest in a logging and monitoring strategy early on before you go to production.
- Gives engineers the expertise to debug and monitor applications.



More time up front to play with tooling

==

Less time learning tooling during prod issues

Logging and Monitoring

- Your existing tooling most likely plays well with Kubernetes.
- Open source is a viable option as well.
- CNCF ecosystem:
  - EFK stack for logging
  - Prometheus and Grafana for monitoring

Containers…

Container Image Best Practices

- Image Sizes

GOAL: Smaller the image, the better

- Less things for an attacker to exploit.
- Quicker to push, quicker to pull.

Container Image Best Practices

GOAL: Don’t rely on :latest tag

- :latest image yesterday might not be :latest image tomorrow
- Instead, you’d want to know what specific version you’re operating with.

Container Image Best Practices

GOAL: Consider using a private registry

- Enterprise concerns for data storage.
- Registry physically closer to your Kubernetes cluster
- Quicker image pulls = faster deployments to Kubernetes
- Consider using the your cloud provider for the registry

Devops

Managed Kubernetes Services

Should I install my own, or use a managed service?

Managed Kubernetes Services

Pros:

- Offload control plane management to the provider.
- Less maintenance headache.
- Spend time working on your apps, and required infrastructure.

Cons:

- Not 100% customizable.
- Hidden costs.

General guidance: Use it, unless you have a non standard usecase.

Cluster Management

Dev/Test/Production clusters strategies?

Cluster Management Strategies

- Two primary strategies in play:
  - Utilize different namespaces in single cluster
  - Utilize different clusters for dev/test/prod

Cluster Management Strategies (Namespaces)

- Single cluster, with multiple namespaces
  - “dev/test/prod”
- Access control via kubeconfig to only have rights to a single namespace.
- Typically used in startups or companies with smaller ops teams.
- Pro: single cluster, so lesser management.
- Con: cluster issues will cause all environments to experience issues.
- Read more: https://kubernetes.io/blog/2016/08/kubernetes-namespaces-use-cases-insights/

Cluster Management Strategies (Separate clusters)

- Multiple unique clusters for separation of concerns
  - Unique dev, test and production clusters
- Easier to implement with managed services (one click)
  - Access control can be implemented in cloud layer, flows down to kubeconfig file.
- Recommended approach for enterprises.
- Cons:
  - More environments to manage

Tagging nodes

Tag your nodes

- Multiple clusters = multiple nodes.
- Be diligent about labeling nodes on creation so that you have better control over your cloud infrastructure.
- Kubernetes has concept of labels, use it!!
- https://kubernetes.io/docs/concepts/overview/working-with-objects/labels/

Pipelines

Pipelines

K8s allows you to build a devops pipeline.

1. Build artifacts
2. Test code
3. Push to registry
4. Optionally Deploy to K8s

$\left.\begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \end{array}\right\}$ Accomplish with CI/CD Tooling

Chance to modernize infra if you haven't already

Cloud Native Enterprise Transformation

How to start?
Where to start?

5 easy steps!

Start small…

Step 1: Get experience with K8s clusters.

Step 2: Take one application or microservice and convert to K8s.

Step 3: Run this application in a production setting.

Step 4: Understand how to manage and firefight.

Step 5: Goto step 2.

Know your teams

- Every organization is different.
- Build your cloud native transformation around your teams.
- Need equal expertise in development, operations and firefighting.
- Organize into development, devops and SRE teams.

- Leverage OCI + Opensource technologies.
- Watch CNCF space because landscape changes (https://www.cncf.io/)

KEEP CALM
AND
KUBE ON

@iteration1