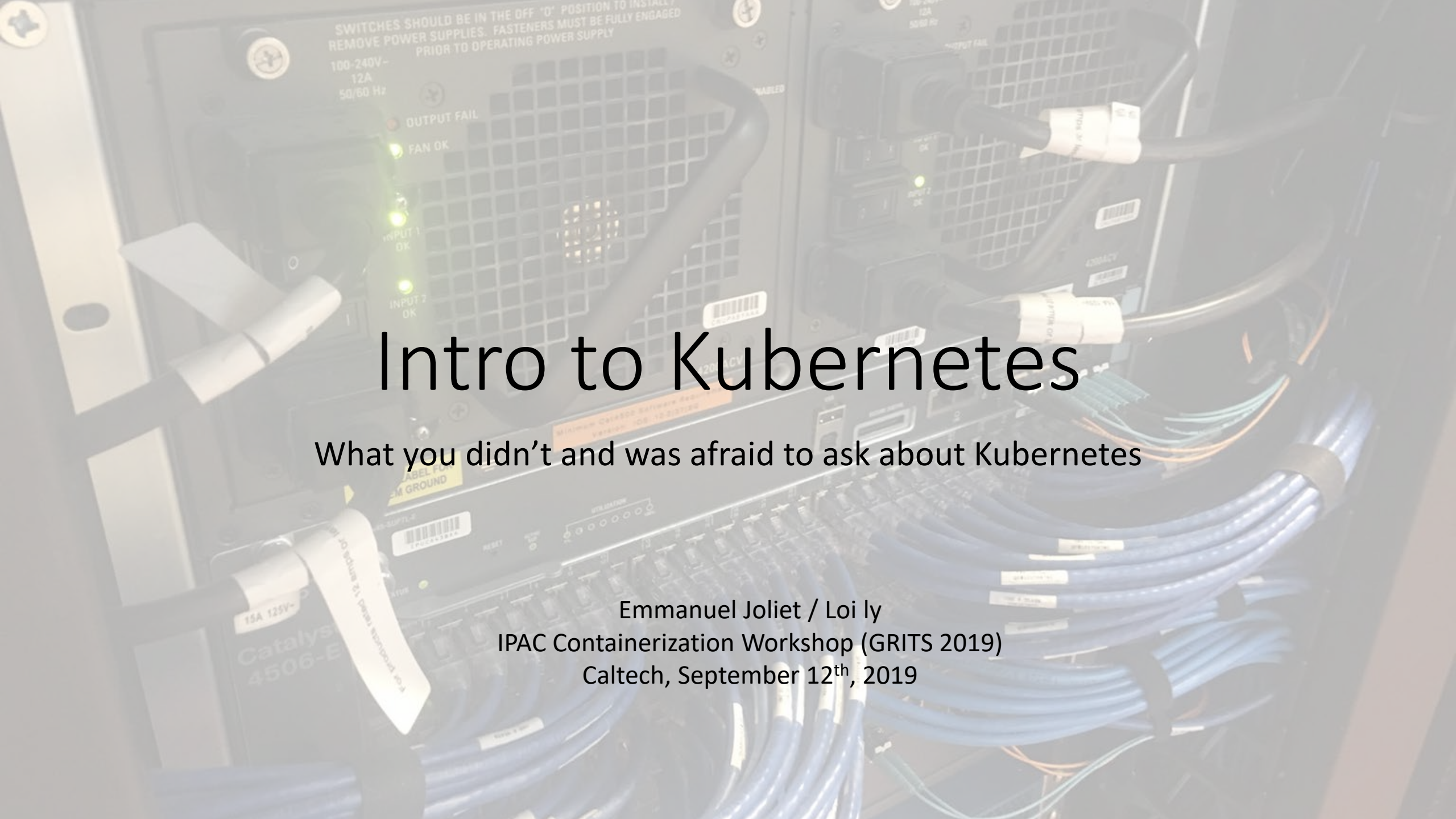


Intro to Kubernetes

What you didn't and was afraid to ask about Kubernetes

Emmanuel Joliet / Loi ly

IPAC Containerization Workshop (GRITS 2019)

Caltech, September 12th, 2019

Problems

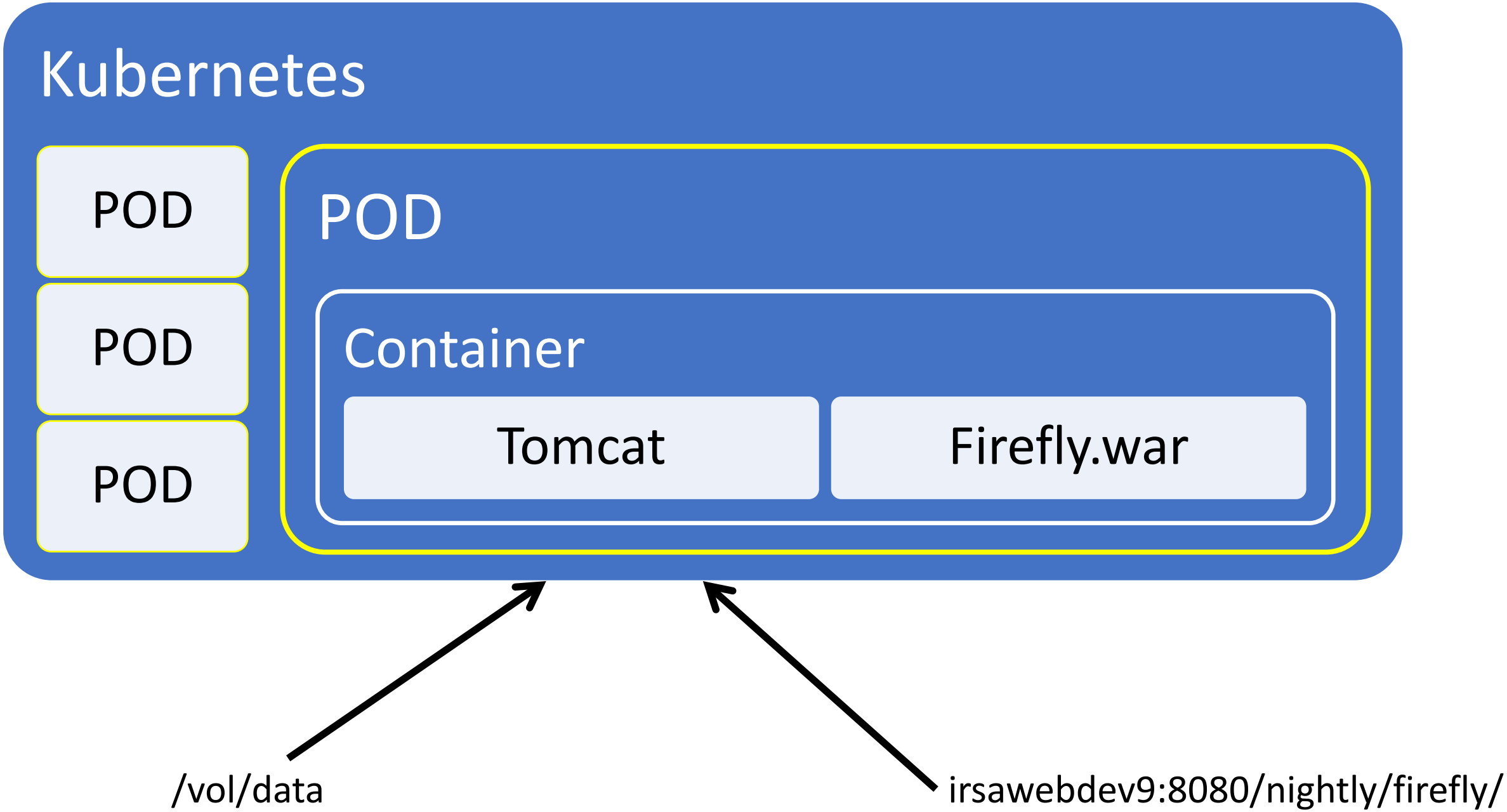
- See virtualization problem and container solution in previous talk(s)
- Now, I have more than one container, what should I do?
- Container is down
- Cloud has changed
- Number of users increased, how do I start more containers?
- Can I do automatic load balancing while I’m having coffee?
- Can I monitor if my container(s) goes down?

Solution

- Docker made popular containers
- Container = scale (lightweight?) and portable (self contained, no external dependencies)
  - portable across clouds and OS distributions (!=VMs)
- Fail-over and load-balancing: Container can be replicated at runtime
- Need an **orchestration** to deal with multiple hosts and containers
- Kubernetes jargon, orchestra is cluster of nodes and master(s)
  - Pod(s) in a node run container(s): storage, network and cpu
  - `Agent` - each node has one to communicate with master node
  - `Pod` = collection of one or more containers
- Efficiently distributing the workload across available resources (VMs!)
- Orchestration can be deployed in local cluster or cloud
- Network policy to avoid exposing all nodes, but only the one that connect outside to protect the rest – Ingress controller (it’s a container, surprise!)

Kubernetes

- Open-source system for automating deployment, scaling, and management of containerized applications
- reduce infrastructure requirements by easily scaling up and down your entire platform, vertically (± hosts resources) and horizontally (± pods)
- Orchestra: what containers run where and when across your system
- From outside, URL to application doesn’t need to change while internal pods changes (for example: software update, system update or balancing/scaling)
- Install local utilities: kubectl and minikube https://kubernetes.io/docs/tasks/tools/install-kubectl/
- Checkout the Kubernetes docs https://kubernetes.io/docs/home/
- Example of usage in IPAC: Firefly development pods following git pull request workflow, each pod deplys a build of a branch – see Jenkins


Kubernetes
POD
POD
POD
POD
Container
Tomcat
Firefly.war
/vol/data
irsawebdev9:8080/nightly/firefly/

Deployment

- Kubernetes uses ‘object’ to specify state and information, using either the API or the client CLI ‘kubectl’
  - `kubectl apply -f deployment.yaml`
- Objects are defined with configuration file in YAML format
- Cluster (nodes running containerized apps) will run and control those objects, typically pods
  - Kubernetes support Docker as container runtime among others, via `kubelet` agent
  - Includes container(s), unique network IP, storage resources
  - Single container is most used model
  - Run on nodes via a controller handling replication and failures
  - Have lifecycle with phases pending, running, etc.
- Expose pods to outside, need clusterIP and Service to (load) balance the pods themselves and have unique address for outside
  - Ingress object or controller
  - Define routing traffic

IRSA UI use case

- Kubernetes in dev environment for UI development
- Docs and files in Firefly github repos
  - https://github.com/Caltech-IPAC/firefly/blob/53618ba5d60c81c2b26d0f3611b0f4384430e1b9/docker/k8s/firefly.yaml
- Jenkins jobs triggers a build, docker image and spinning up a pod with the container
  - Pull request branches are tested using a pod
  - Unique URL is exposed irsawebdev9

Kubernetes: v1.8.5
Docker: v17.03.2
VMs: irsawebdev9,10,11,12 Debian 8
irsawebdev9 -> master
others -> nodes (workers)

```
irsadmin@irsawebdev9:~$ kubectl get nodes
NAME          STATUS   ROLES    AGE   VERSION
irsawebdev10  Ready    <none>   10d   v1.15.3
irsawebdev11  Ready    <none>   10d   v1.15.3
irsawebdev12  Ready    <none>   10d   v1.15.3
irsawebdev9   Ready    master   10d   v1.15.3
```