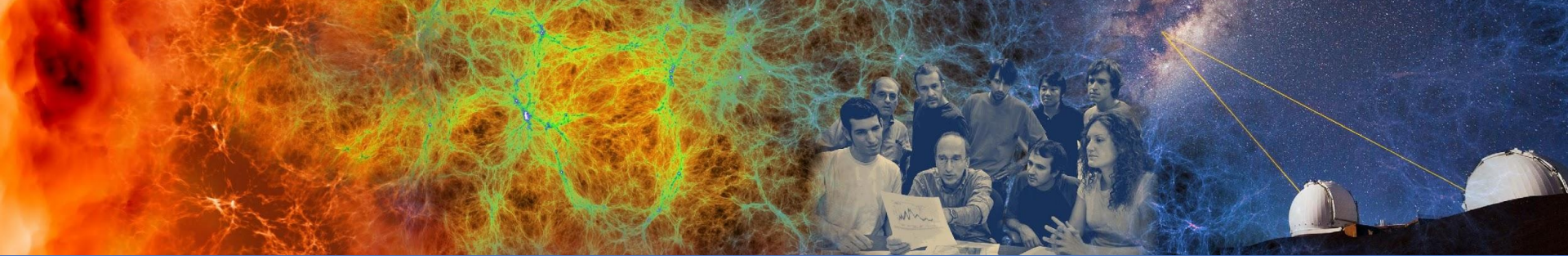


# Spin is a Powerful System...

...and with great power comes great responsibility!

- **Keep software updated; fix vulnerabilities promptly.**
  - *NERSC scans regularly to find problems quickly.*
- **Encrypt anything accessible over the network.**
  - *These are strict DOE and DHS requirements!*
- **Produce logs to stdout/stderr.**
  - *This is Docker convention anyway.*

Don't worry. Spin helps make these best practices easy!



# Concepts and Terminology

# Why Do We Need Spin?

**Your project is more than batch jobs and data files; it's science gateways, databases, and other services.**

Spin is a supported platform designed to help:

- *Cloud-style flexibility*
- *Create new apps yourself on demand*
- *Redundancy / uptime (99% in 2020)*
- *Direct access to HPC file systems and networks*

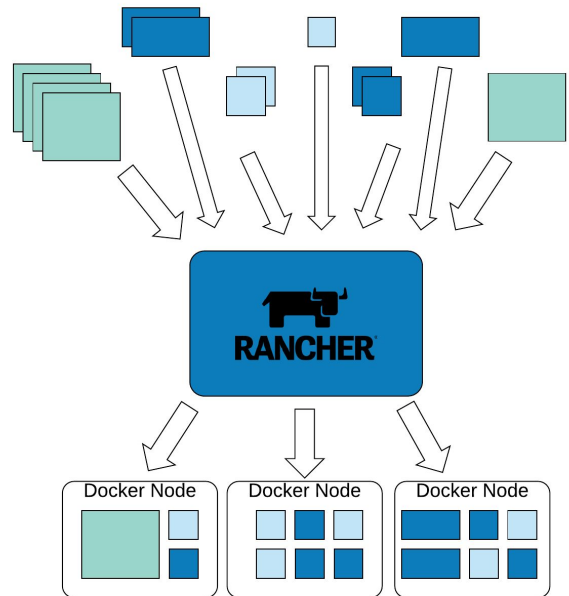
# Docker, Kubernetes, and Rancher

**Spin is based on the Rancher *orchestration* system, which is built on Docker and Kubernetes.**

***How do they all fit together?***

- Docker is great for just you on a laptop.
- For lots of applications, you need a whole Kubernetes *cluster*.
- For lots of projects, each with lots of applications, we need *orchestration*.
- With Rancher orchestration, you get *virtual private* access to the *multiple* Kubernetes clusters running in Spin.

Without orchestration, a pool of servers and no coordination for users



Managed and assigned to Docker nodes, enabling holistic management, failover, service ownership.

# (Some of the) Terminology

**Container image:** blueprint for a container; like a tarball

**Container:** running instance of an image; like a process

**Image Registry:** versioned repository for container images

**Pod:** one or more very-closely-coupled containers

**Workload:** set of parameters and rules that define how to create a *particular* pod

**Deploy:** create a workload

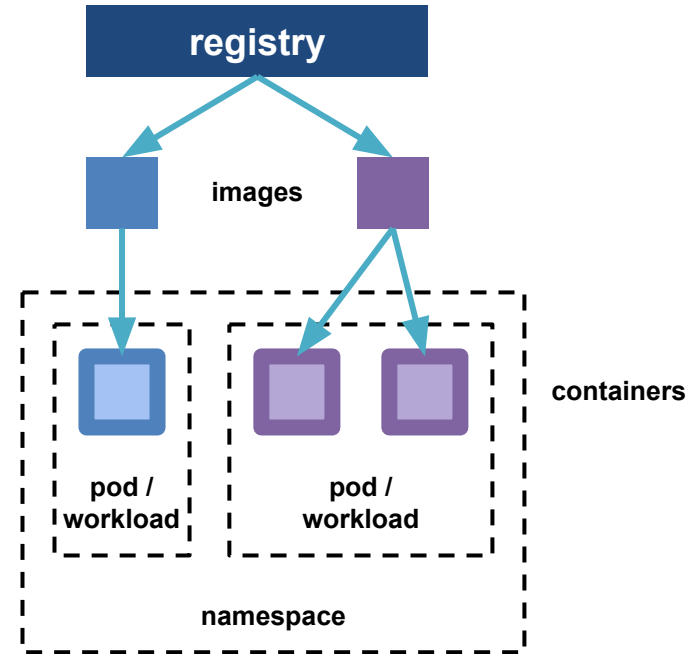
**Ingress:** proxy that allows a workload to be accessible on the network using a DNS name

**Namespace:** group of workloads (often for interoperation)

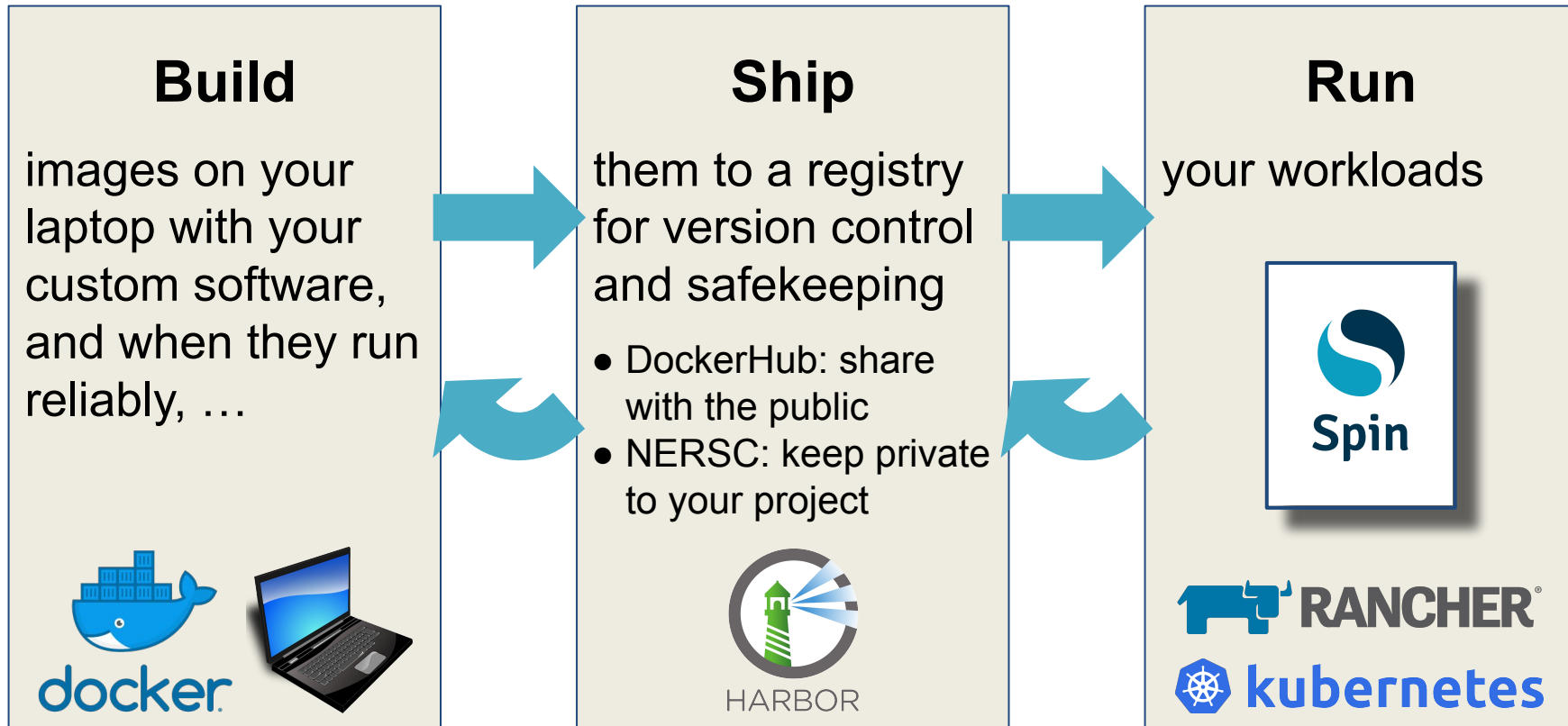
**Project:** group of workloads, namespaces, ingresses, etc for access control; corresponds to a NERSC project

**Kubernetes:** container *scheduling* system to run it all

**Rancher:** *orchestration* system for Kubernetes clusters



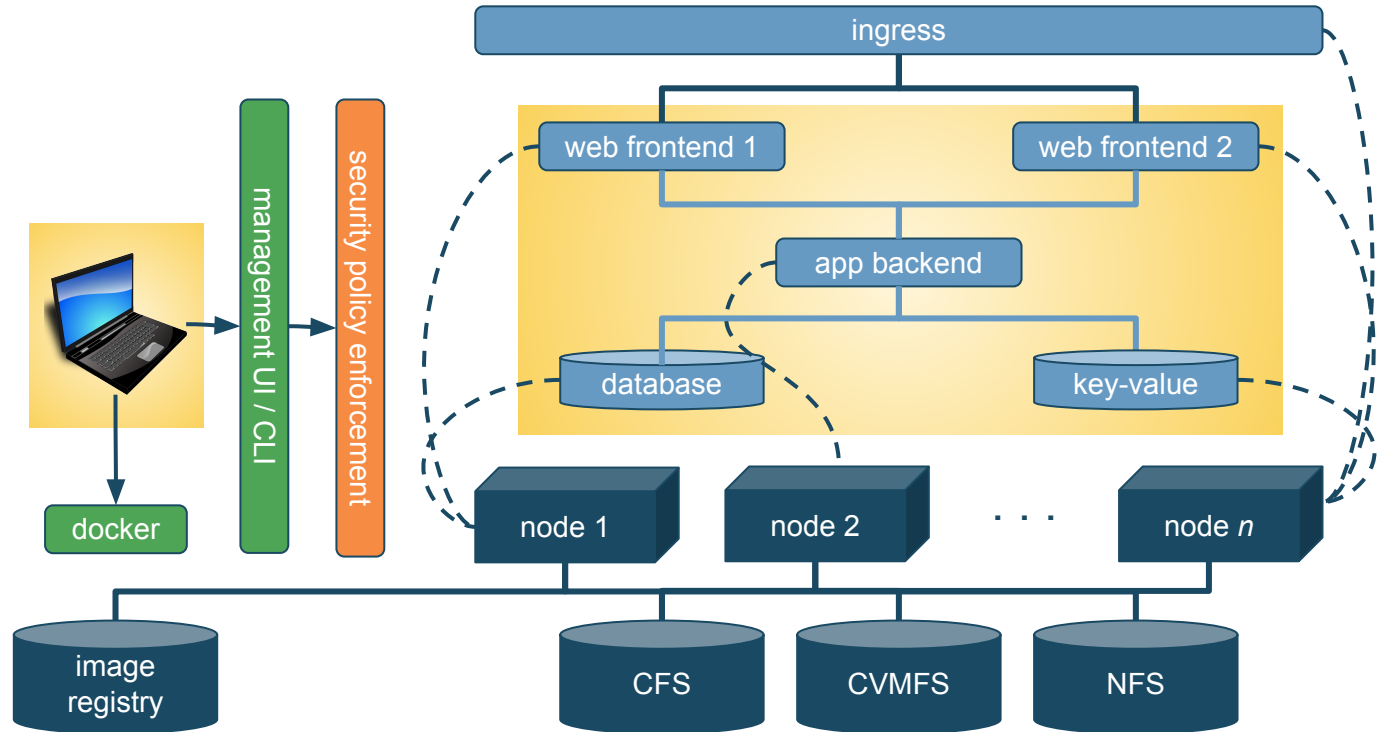
# Canonical Development Workflow



# High-Level Spin Architecture

**Yours to  
manage**

**NERSC  
handles  
the rest!**



# Interactive Exercises: Let's Create an App!

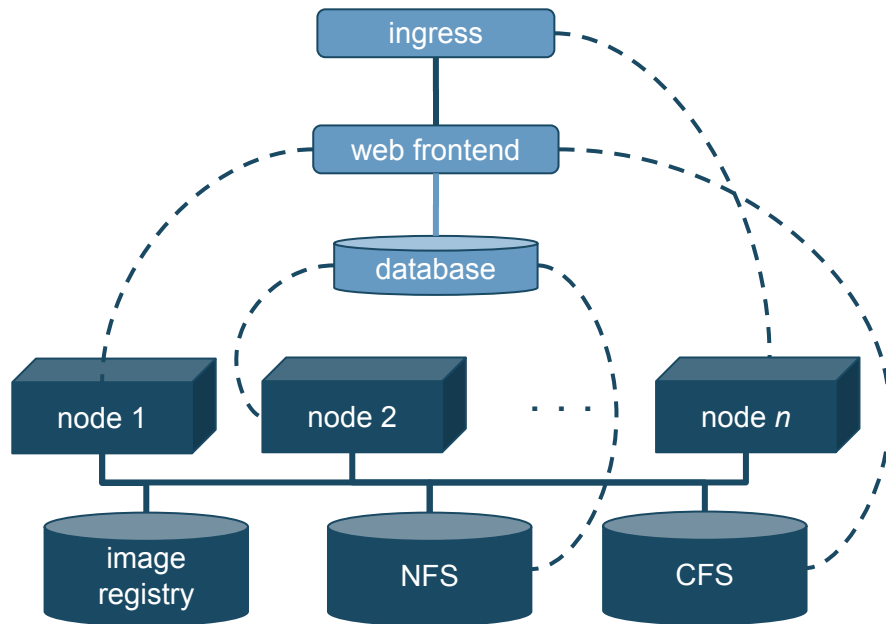
## Our example app:

- Python-based
- Uses static files in CFS
- Database backend

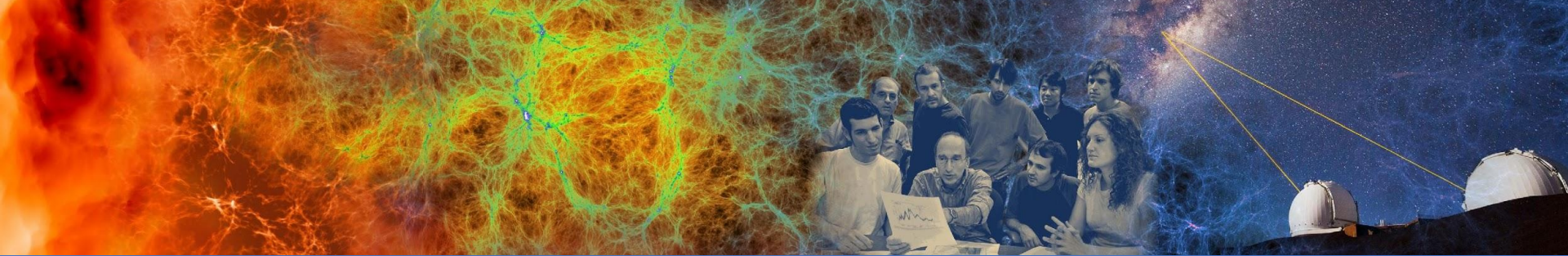
***We will build the app from the bottom up, database first.***

## Along the way, we will

- Use variables and config maps to customize behavior
- Attach storage
- Store passwords securely
- Make it available on the network







# Exercise 1: Create a Database

# Exercise 1: Create a Database

- Databases often underlie web apps, so let's start there.
- In Spin, you can access an external database or create your own, as we'll do now.
- We recommend using stock images from DockerHub for MongoDB, MySQL, PostgreSQL, Redis, and others.
  - Frequently updated, easy to customize...less work!
- Look at the README: [https://hub.docker.com/\\_/mysql](https://hub.docker.com/_/mysql)
  - Customize by setting variables; no custom image needed

# Watch an Example

The screenshot shows the Rancher2 web interface in a web browser. The address bar displays the URL: `rancher2.spin.nersc.gov/p/c-fwj56:p-6q5dd/workloads`. The interface has a dark blue header with navigation tabs: **development** (selected), **spinup**, **Resources**, **Apps**, **Namespaces**, **Members**, and **Tools**. A dropdown menu is open under the **development** tab, showing a list of clusters: **Global** (with a search bar), **development** (Active), **local** (Active), **spinup** (Active), **ssg** (Active), **stefanl** (Active), **System** (Active), and **usg** (Active). The main content area displays a table of workloads. The table has columns for **Workload**, **Image**, and **Scale**. The workloads listed are: **spintst4** (Active), **spinup** (Active), **ssg** (Active), **stefanl** (Active), **System** (Active), **usg** (Active), **paradise-iverilog3** (Inactive, with a message: "ReplicaSet 'paradise-iverilog2-788b4bb6c7' has timed out progressing." and "Admission webhook 'validating-webhook.openpolicyagent.org' denied th..."), and **genimg** (Active). The **genimg** workload is shown with a scale of 1. The interface also includes buttons for **Redeploy**, **Download YAML**, **Delete**, **Import YAML**, and **Deploy**. A search bar is located in the top right corner of the main content area.

<https://rancher2.spin.nersc.gov/g>

# Try It Yourself!

1. Log in to <https://rancher2.spin.nersc.gov>.
2. Under Global, select the **development** cluster, then select a project.
3. At the top right, click **Deploy** and enter  
Name: db  
Image: mysql:5
4. Click **Add to a new namespace** and enter something unique. *Note: underscores (\_) are not allowed in namespace names!*
5. Expand **Environment Variables** and enter  
MYSQL\_DATABASE = science  
MYSQL\_USER = user  
MYSQL\_PASSWORD = pw  
MYSQL\_RANDOM\_ROOT\_PASSWORD = yes  
TZ = US/Pacific *or other timezone*
6. At the bottom right, click **Show advanced options** and expand **Security & Host Config**.
7. Under **Add Capabilities**, select  
CHOWN, DAC\_OVERRIDE, FOWNER,  
SETGID, SETUID
8. Under **Drop Capabilities**, select  
ALL
9. Click **Launch** and watch the pod start up.
10. Open the (⋮) menu and select **Execute Shell** to create a table:  

```
# mysql -u user -D science -p
mysql> create table t(n integer);
```

# Discussion

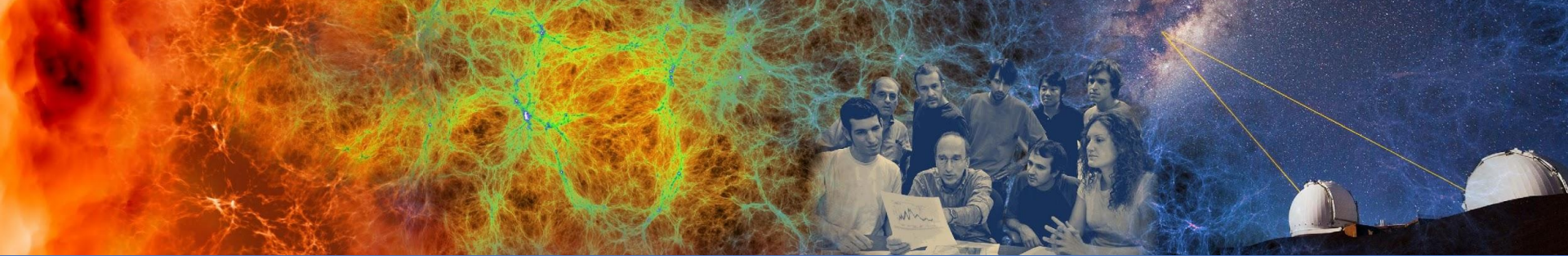
- Terminology: You **deployed** a new **workload** in a new **namespace** in a **project** on the *development* **cluster**. It has one **pod** running one **container** based on the stock MySQL **image**.
- Good stock images make life easy, but be prepared to
  - Read the READMEs for how to set variables
  - Look inside with `docker exec -it image /bin/bash`
- Shell access is easy; no ssh daemon required.

# Discussion

Capabilities are root powers; Spin allows them selectively.

Later, we'll discuss how capabilities are limited even further when using global file systems.

Capability	Meaning
CHOWN	Change the owner of files and directories
DAC_OVERRIDE	Override file permissions
FOWNER	Override owner permissions
NET_BIND_SERVICE	Open network ports numbered < 1024
SETGID	Change the group of a running process
SETUID	Change the user of a running process



## Exercise 2: Add a Secret

## Exercise 2: Add a Secret

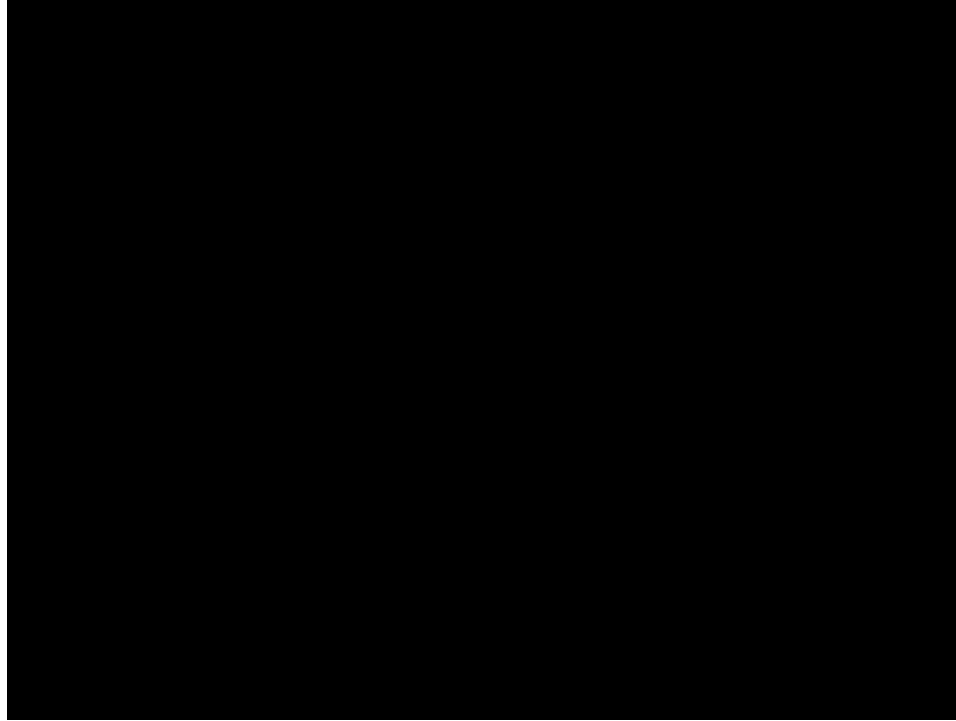
- The password seems a little too exposed. Is there a better way to handle things I want to keep secret?
- How can I see what's happening with my service? How can I see logs?
- What happens when I change a workload? Are there any gotchas I should watch out for?



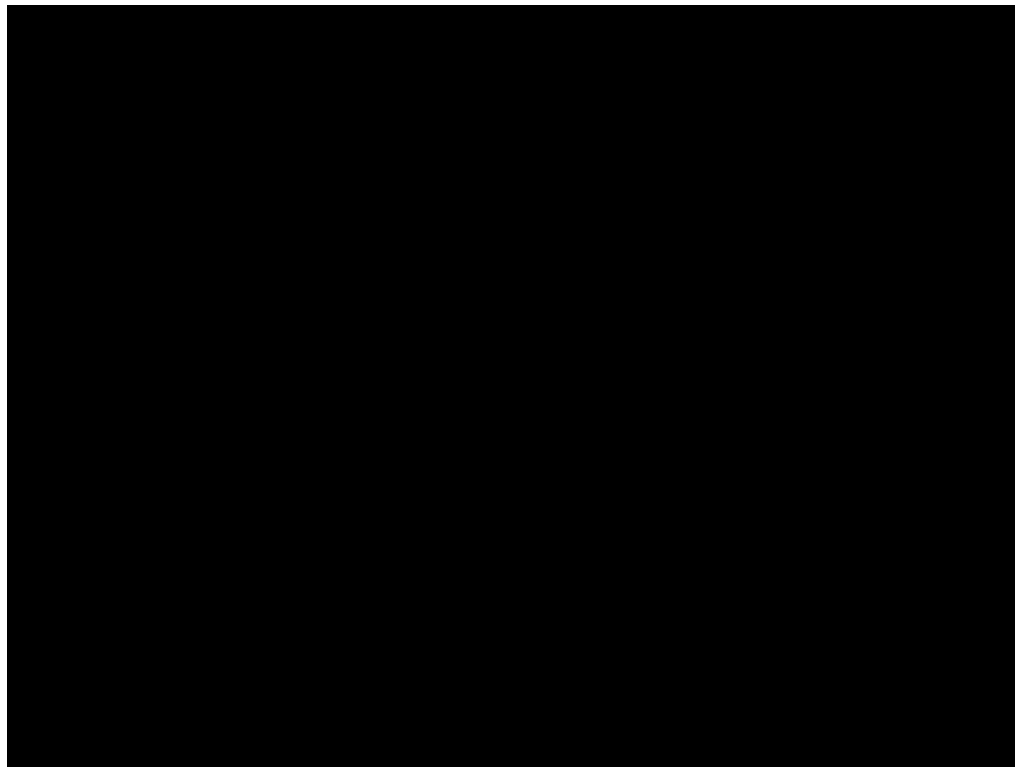
# Watch an Example

- Create a secret
- Use a secret
- Look at the logs
- See what happened to the previously created table

# Create and use a secret



# Look at the logs; data is ephemeral



# Try It Yourself!

1. Select **Resources > Secrets** and click **Add Secrets**.

Select: Available to a single namespace  
Select the namespace in the drop down

2. **Set Values**

Name: `db-password`

Key: `password`

Value: `<make-something-up>`

3. Click **Save**

Create the secret

1. Click on **Resources > Workloads**, open the (⋮) menu to the right of your workload, and select **Edit**.

2. Expand **Volumes**; click **Add Volumes**; select **Use a Secret**.

3. From the **Secret** drop-down, choose `db-password`.

4. Check **Select Specific Keys**; from the **Key** drop-down, choose `password`. Under **Path**, enter `password`.

5. Set **Mount Point** to `/secrets`.

6. Click **Save**.

Attach the secret

5. Use **Exec Shell** to look at the results

```
# cat /secrets/password
```

6. Click **Edit**, expand **Environment Variables**, and replace `MYSQL_PASSWORD: pw` with `MYSQL_PASSWORD_FILE: /secrets/password`

7. Click **Save**

Use the secret

8. Click on the database Pod, open the (⋮) menu, and select **View Logs** for the running Pod.

9. Use **Exec Shell** again and use the new password to connect to MySQL

```
# mysql -u user -D science -p
```

Test the secret

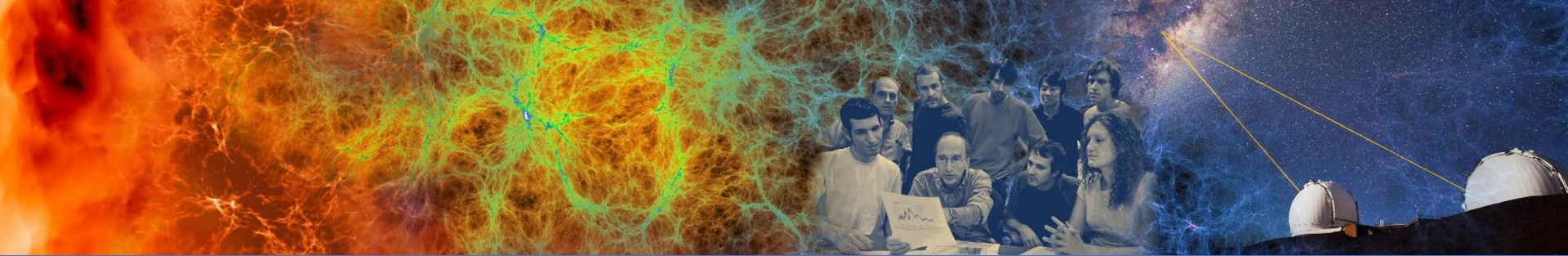
10. Notice: starting a new pod re-initiated the database!

```
mysql> show tables;
```

App Workload

# Discussion

- Secrets are a good way to manage and protect passwords, tokens, etc.
- Secrets can be scoped to a project or a namespace
- View Logs can help you understand and monitor your deployments
- Containers are ephemeral unless you use other storage methods (next)



# Exercise 3: Add NFS Storage

## Exercise 3: Add NFS Storage

Remember, Docker containers are ephemeral. Your changes go away when a new container is started. Persistent storage can allow you to make changes stick.

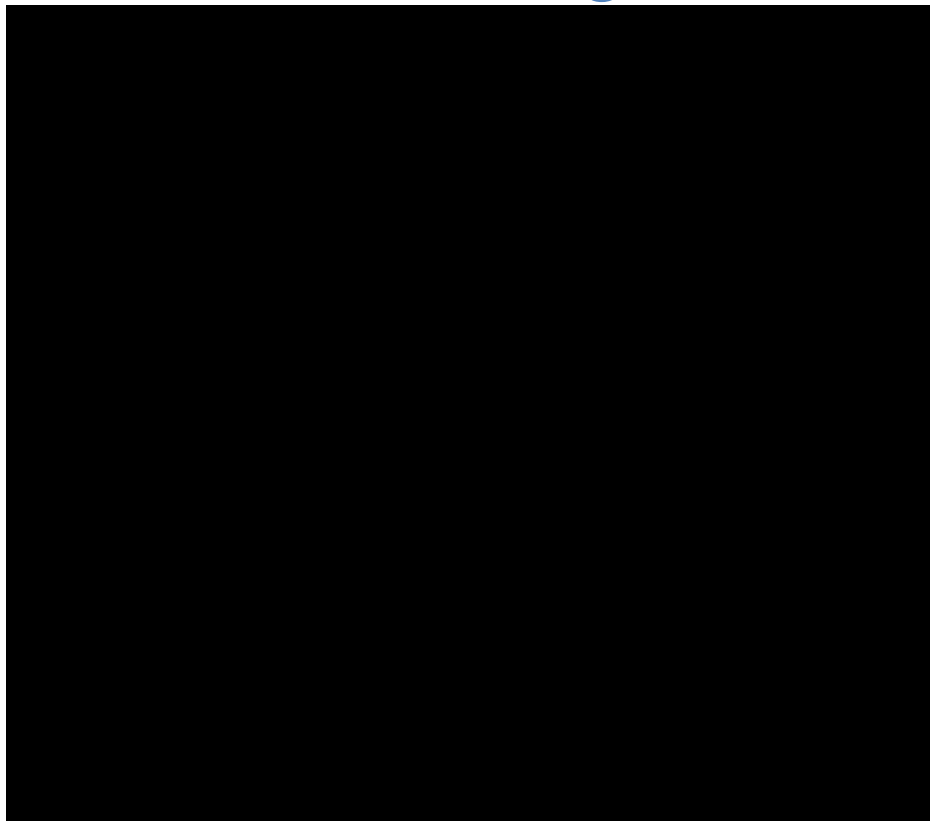
### NFS Storage in Spin is

- High performance
- High availability (same as Spin itself)
- Mountable into >1 workload (even across namespaces)
- Mounted only on Spin (not other NERSC systems)

Another option: NERSC Global Filesystems (coming up)

# Watch an Example: Add NFS Storage

In this video example, you will learn how to set up a Volume Claim so that updates to your database are saved.





# Try It Yourself!

## Set up the NFS Volume

1. In your project, open the workload for which you want to add storage.
2. Open the (⋮) menu and click Edit.
3. Open the Volumes accordion.
4. Click Add Volume and select “Add a new persistent volume (claim)”.
5. Fill in a name for the volume. Under Storage Class, select “nfs-client”. Set the Capacity to 1 GiB. Click Define.
6. Fill in the mount point where it should appear in the container, `/var/lib/mysql`.
7. Leave Sub Path in Volume blank.
8. Click Save at the bottom of the page.

## Test changes to the database

1. Open the (⋮) menu, select **Execute Shell**, and create a table like you did before:

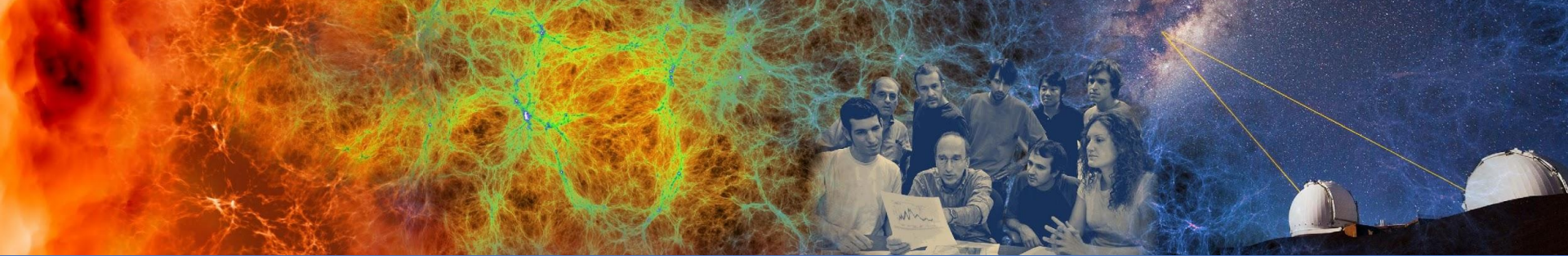
```
# mysql -u user -D science -p
mysql> create table t(n integer);
```

2. “Edit” the container: open the (⋮) menu and select Redeploy (or select Edit, then Save, which has the same effect).
3. Run **Execute Shell** again and do a “show tables;” to see that your changes persist:

```
# mysql -u user -D science -p
mysql> show tables;
```

# Discussion

- NFS Storage enables data to persist across container instances.
- They allow persistent, performant, read-write storage.
- They are not mounted elsewhere, so you may need to set up a utility container for backups, permission changes.
- They are best used when the data are not needed across NERSC systems.

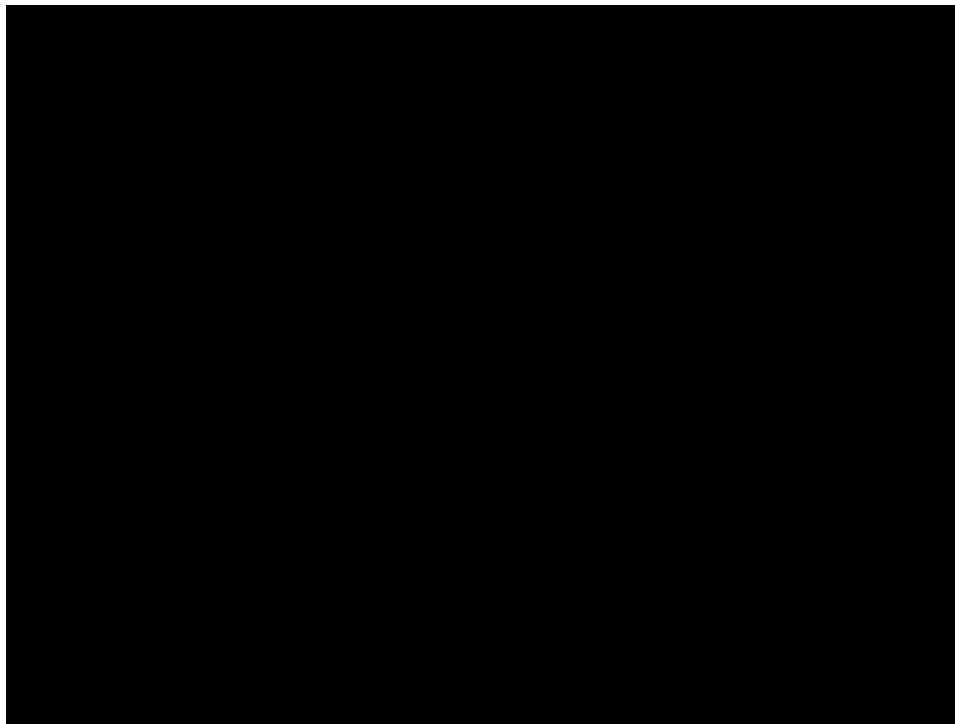


## Exercise 4: Add a Web Front-end and CFS

# Exercise 4: Add a Web Front-end and CFS

- Most use cases for Spin are apps that expose data on CFS or functionality at NERSC over the web.
- We've created one in a Docker image that uses:
  - Flask to handle HTTP requests, routing, responses
    - Pretty simple galaxy cluster gallery app
  - Config map for setting some environment variable
  - Database for content and metadata
    - Stored on NFS
  - Image files for web front-end to serve up
    - Stored on CFS

# Watch an Example



# Try It Yourself!

1. **Resources > Config** then **click** “Add Config Map” to see the “Add Config Map” console, then **set**:

Name: <your config map>  
Namespace: <your namespace>

2. **Set** “Config Map Values” key/value pair:

banner\_message = <something hilarious>

3. **Click** “Save” button

**Config Map**

1. **Resources > Workloads** then **click** “Deploy”

Namespace: <your namespace>

to see “Deploy Workload” console, at top **set**:

Name: app  
Docker Image: registry.nersc.gov/spinup/galaxies:latest

2. **Expand** “Environment Variables” panel to configure 2 variables:

**Click** “Add Variable” and **set**:

MYSQL\_PASSWORD\_FILE = /secrets/password

**Click** “Add From Source,” and **set**:

Type: Config Map  
Source: <your config map>  
Key: banner\_message  
Prefix or Alias: BANNER\_MESSAGE

3. **Expand** “Volumes” panel to configure 2 volumes:

**Open** “Add Volume” dropdown

**Select** “Bind-mount a directory from the node” and **set**:

Path on the Node:

/global/cfs/cdirs/mpccc/rthomas/spin-demo/static

The Path on the Node must be: An existing directory

Mount Point: /srv/static

Read-Only: [✓]

**Open** “Add Volume” dropdown to add a new volume

**Select** “Use a secret,” **set**:

Secret: db-password Select Specific Keys [✓]

Key: password

Path: password

Mount Point: /secrets Read-Only [✓]

4. **Click** “Show advanced options” to see more panels, **expand** “Command” panel, **set**

User ID: <user ID>

Filesystem Group: <group ID>

Use **id** on Cori to find these values.

5. **Expand** “Security & Host Config” panel and **set**:

Run as Non-Root: Yes

Add Capabilities: NET\_BIND\_SERVICE

Drop Capabilities: ALL

6. **Click** “Launch” button

**App Workload**

# Discussion: App

- Where did the image come from?
  - Built image locally
  - <https://github.com/NERSC/spin-docker-compose-example>
    - Contains the app.py code, Dockerfile, entrypoint, etc.
    - Image data included too though this is for demonstration only
  - Push to registry.nersc.gov/<project>/<image-name>:<tag>
- How was the database initialized?
  - “Before first request” Flask decorator:
    - Connect to the database
    - Try to create the data table and fill with data
    - Not a robust error check here, it’s a demo
    - Do this because the app container might restart

# Discussion: Global File Systems

- **Using global file systems such as CFS triggers stricter security!**
  - Set User ID to yourself or a collab user;
  - Set Filesystem Group to one you belong to  
*Otherwise, projects' files could be exposed*
  - Only one capability allowed: NET\_BIND\_SERVICE  
*Otherwise, file system permissions could be bypassed*
- **Set o+x permissions from file system root to mount point**
- **Best practices**
  - use read-only access unless you *specifically* need read/write
  - mount as deep into the path as possible
  - use collab users
  - use setgid (chmod g+s) and a group-friendly umask (eg, 007)



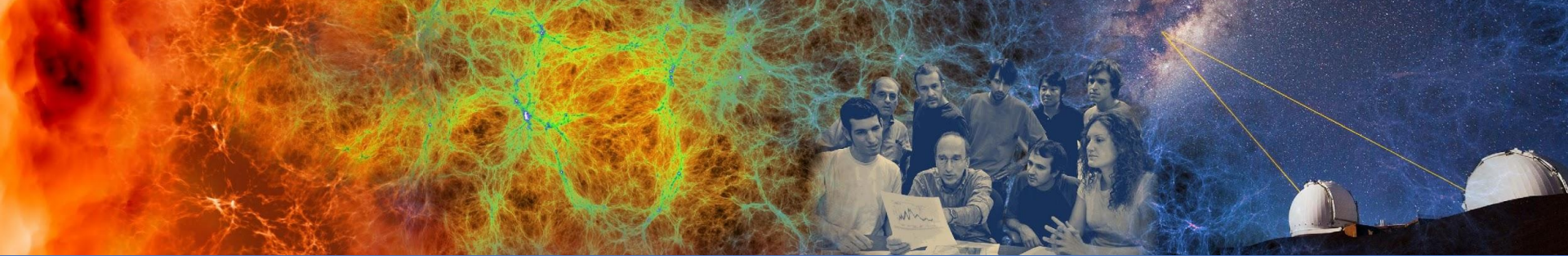
# Discussion: Storage Options

Storage Type	Persistent	On HPC	Size	Best Use
<i>Global File Systems (Homes, CFS)</i>	✓	✓	O(quota)	sequential
<i>NFS</i>	✓		O(10GB)+	random
<i>CVMFS (read-only) always mount at root!</i>	✓	✓	n/a	CERN software
<i>in-container</i>			O(1GB)	temporary

# Discussion: Storage Options

Storage Need	Best Option
Data produced by compute jobs and used by science gateway	Global file system
Static web content or config files that require occasional updates	Global file system*
Web service access logs to analyze and save for record-keeping	Global file system*
Database tablespace or key-value backing store files	NFS
Static application code and web style sheets	in-container
Small, ephemeral application cache files	in-container

***What other examples? What are some exceptions?***



# Exercise 5: Networking & Ingress

# Exercise 5: Networking (Internal Overlay)

Containers **inside cluster** communicate over an **overlay network**

- Internal IP range: 10.42.0.0-10.42.255.255
  - Addresses are assigned randomly to your application
- Internal DNS & service discovery handled by CoreDNS
  - The **app** container already knows about the **db**:

`db.<namespace>.svc.cluster.local`

- Network policies ensures network access only within your project
- Performance between containers: 1-5 Gbps
- Takeaway: Once you get used to it, it just works

# Exercise 5: Networking (External & DNS)

Great! My web app is running. But how do I access it?

- **Ingress:** Enables external access to a web app
  - Maps a Spin IP address (public) to your application
  - HTTP/HTTPS-only (Ports 80 & 443)
- **DNS:** Maps a hostname to the public IP
  - Hostname will stay the same and will always point to a working IP address
    - Spin has multiple public IPs. IP of your ingress will sometimes change-- don't rely on it!
  - Must have one host which follows hostname convention:
    - `<name>.<namespace>.development.svc.spin.nersc.org`
      - Notice: nersc.org, **NOT** nersc.gov!
  - Friendly hostnames like `www.cosmosgallery.org` covered in an upcoming slide
- Access for Non-HTTP/HTTPS services is coming in the future

# Exercise 5: How an Ingress Works

- Terminology: An **ingress** exposes the application to a public network, & works with **DNS** to direct traffic to your application
  - Ingress is built on Nginx
- External DNS record is requested when the ingress created, Removed when ingress is deleted
- Behind the scenes (For those familiar with Kubernetes already)
  - Rancher automatically creates a Kubernetes **service** to connect the ingress & workload, but this is not shown within the UI

# Watch an Example: Add an Ingress

Rancher

Welcome to nginx!

development stefanl

Resources Apps Namespaces Members Tools

## Add Ingress

Name [Add a Description](#) Namespace [Add to a new namespace](#)

lb stefanl

### Rules

☐ Automatically generate a `.xip.io` hostname

☒ Specify a hostname to use

☐ Use as the default backend  
Ingress controller does not support default backend

Request Host  
lb.stefanl.development.svc.spin.nersc.org

Target Backend [+ Service](#) [+ Workload](#)

# Try It Yourself!

1. Start in **Resources > Workload**
2. Click **Load Balancing**, then **Add Ingress**
3. **Set these values**

Create the ingress

**Name:** `lb`

**Namespace:** `<Namespace from previous exercise>`

4. Click **Specify Hostname to use** and add `lb.<namespace>.development.svc.spin.nersc.org`
5. Scroll down to **Target Backend** (The **Workload** type is selected by default) & add these values

**Path:** Leave blank

**Target:** `app`

**Port:** `5000`

6. Click **Save**

Use the ingress

You are back at the **Load Balancing** screen

1. Wait for **State** to change from ***Initializing*** to ***Ready***
2. Wait for DNS to propagate to the LBL/NERSC and other DNS servers (**Usually 1-5 minutes**)
3. Access your app at: `http://lb.<namespace>.development.svc.spin.nersc.org`



# Exercise 5: Add a Friendly Hostname

- Examples: **myscience.lbl.gov** or **www.cosmosgallery.org**

1. Use LBL's DNS service or your favorite DNS provider

a. Create a **DNS alias** (CNAME), point your hostname to:

**`lb.<namespace>.development.svc.spin.nersc.org`**

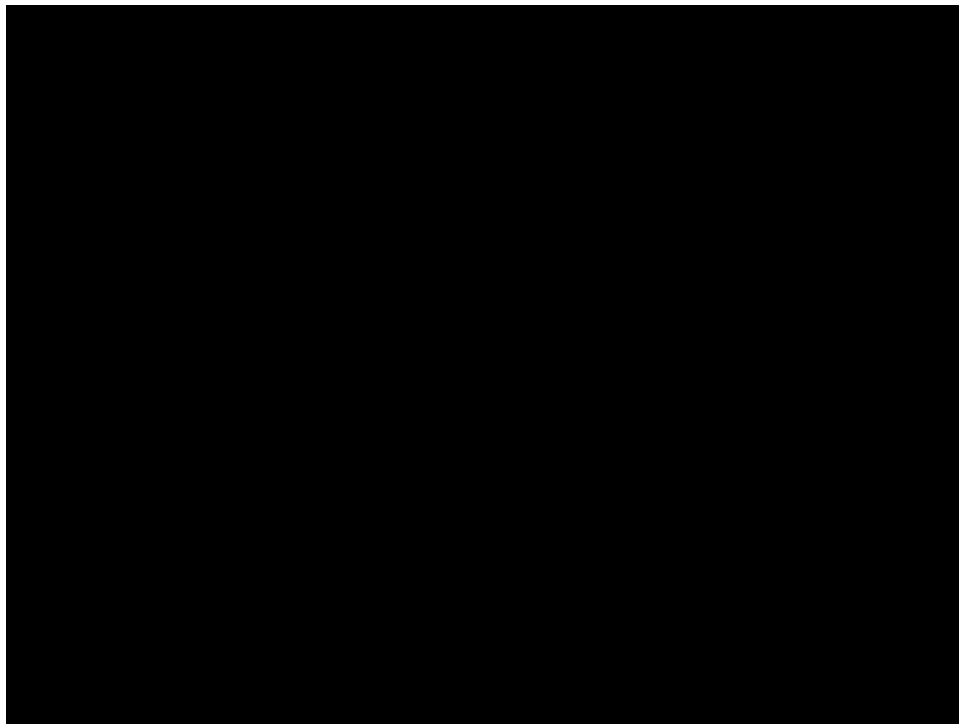
2. Configure Ingress to accept traffic destined for that hostname:

a. In your Ingress -> Add Rule

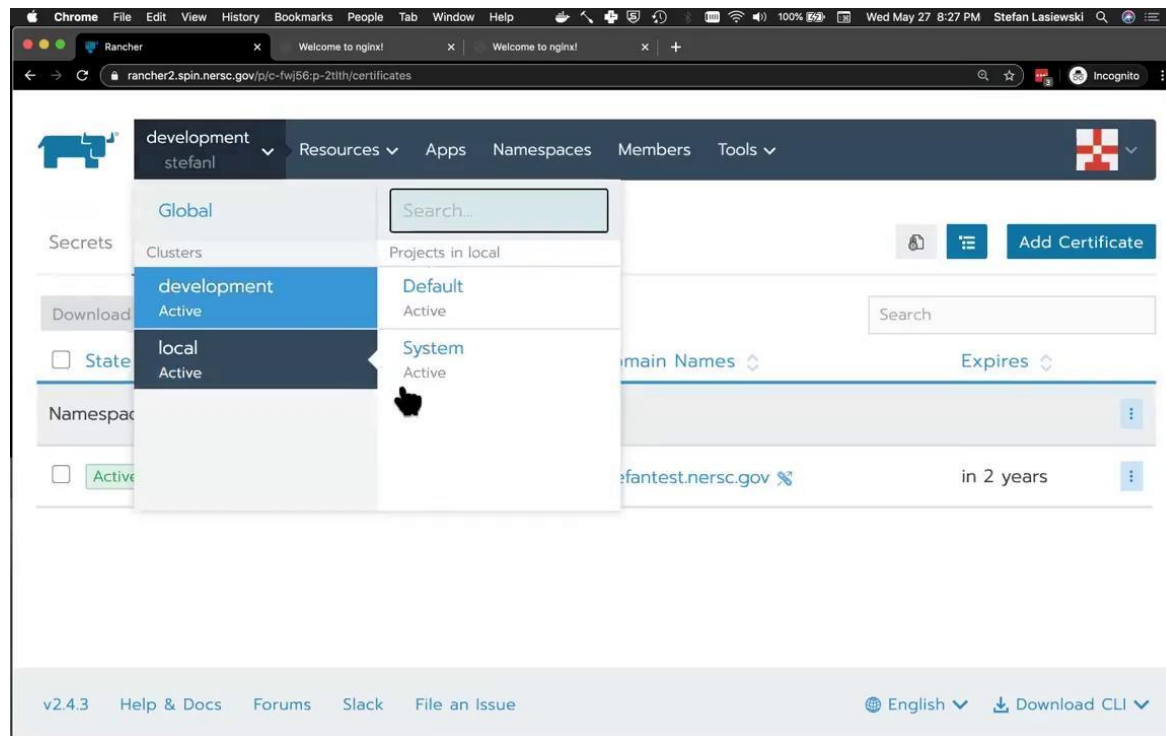
b. Add the friendly hostname as a second "rule"

- For HTTPS, the hostname **must** match name in certificate (See next slide)

# Watch an Example: Add a Friendly Hostname



# Watch an Example: Add a TLS Certificate




# Try It Yourself!

## Friendly Hostname

1. Get a CNAME entry from your DNS provider that points at your ingress. For instance:

```
<something>.myproject.org ->  
lb.<namespace>.development.svc.spin.nersc.org
```

2. When it is ready (hours or days later), navigate to **Resources > Workload** in Rancher.
3. Click **Load Balancing**, then the  icon next to your ingress, and select **Edit** from the dropdown
4. Click **Add Rule**.
5. Select **Specify a Hostname to Use** and enter the CNAME. Do not alter the existing rule.
6. Select the same **Target** workload and **Port** as in the existing ingress rule, then click **Save**.

## SSL/TLS (HTTPS)

1. Get a TLS/SSL certificate from your provider. There are many tutorials on how to do this.
2. Navigate to **Resources > Secrets**, click the **Certificates** tab, then click **Add Certificate**.
3. Enter a meaningful **Name** and select a **Scope**. We don't recommend selecting all namespaces.
4. Upload your **Private Key** and **CA Certificate** using "Read from File" buttons and click **Save**.
5. Navigate to **Resources > Workloads** and then the **Load Balancing** tab.
6. **Edit** the ingress, open the **SSL/TLS Certificates** accordion, select the certificate from the list, and **Save**.

# Discussion: DNS Gotchas

- Wait 2-3 minutes for DNS name to get pushed to the NERSC+LBL DNS servers and propagated to the internet
- Reusing a hostname? Watch out for DNS caching on your Mac and in Chrome (and in your workplace network)!

# Discussion: HTTPS and Certificate Gotchas

Many certificates contain a **certificate chain**:

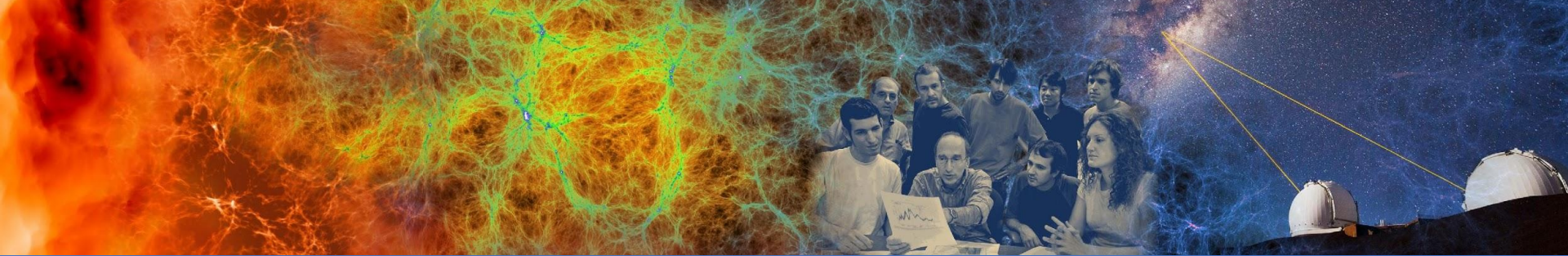
- A certificate for your hostname
- Certificates for the Certificate Authority used to sign your certificate (InCommon, Go Daddy, etc)

The certificate for your hostname must be **listed first in the file**. The key will be checked against the **first certificate only**.

- This type of failure is not obvious
  - Kubernetes will serve a "Kubernetes Default" certificate if the hostname does not match.
- If your certificate doesn't need a chain, you're good!

# Discussion: Non-HTTP Services (DBs, etc)

- Non-HTTP services not supported on Spin yet
  - Kubernetes community still trying determine best solution
  - Many half-solutions, alpha-, beta- software
  - This will improve as Kubernetes matures
- Solutions coming soon to Spin:
  - **Load Balancer (Not Ingress):** direct traffic from NERSC hosts to your application in the Spin cluster. For example:  
  
`db.<myns>.development.svc.spin.nersc.org:32767 -> myns/db:3306`
  - Security is important here!



# Viewing Logs and Performance Data



# Viewing Logs

Log Type	Content	Where	Best Use
<b>Container</b>	All stdout and stderr from container processes	<b>Workload page:</b> expand <b>Pods</b> , select <b>View Logs</b> under (⋮) menu next to pod. <b>Pod page #1:</b> select <b>View Logs</b> under (⋮) menu in top right <b>Pod page #2:</b> expand <b>Containers</b> , select <b>View Logs</b> under (⋮) menu next to container	Application problem, but container runs  Container produces error at startup, exits, and restarts
<b>Pod Events, Pod Status</b>	Scheduler activity (start, stop, scale)	<b>Workload page:</b> expand <b>Events</b> <b>Pod page:</b> expand <b>Events</b> and/or <b>Status</b>	Workload will not start or scale at all  Container restarts continuously

# Performance Analytics

## Rancher provides live Grafana plots of Kubernetes Resource Metrics:

- CPU Utilization
- Memory Utilization
- Network packets and throughput
- Disk throughput

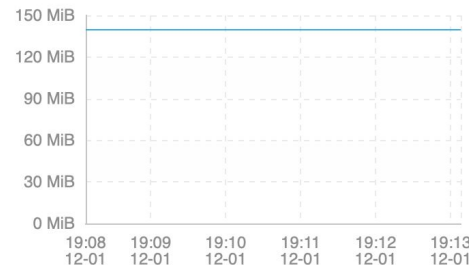
### Where:

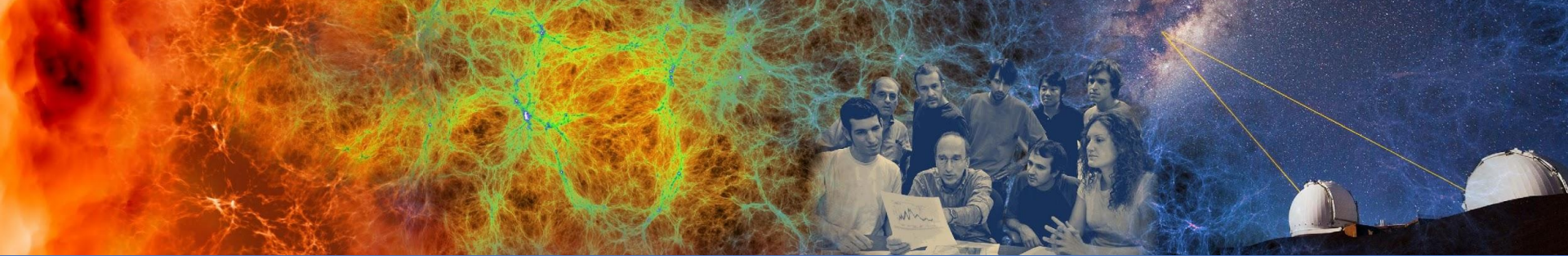
- **Workload page:** expand **Workload Metrics**
- **Pod page:** expand **Pod Metrics**

CPU Utilization



Memory Utilization



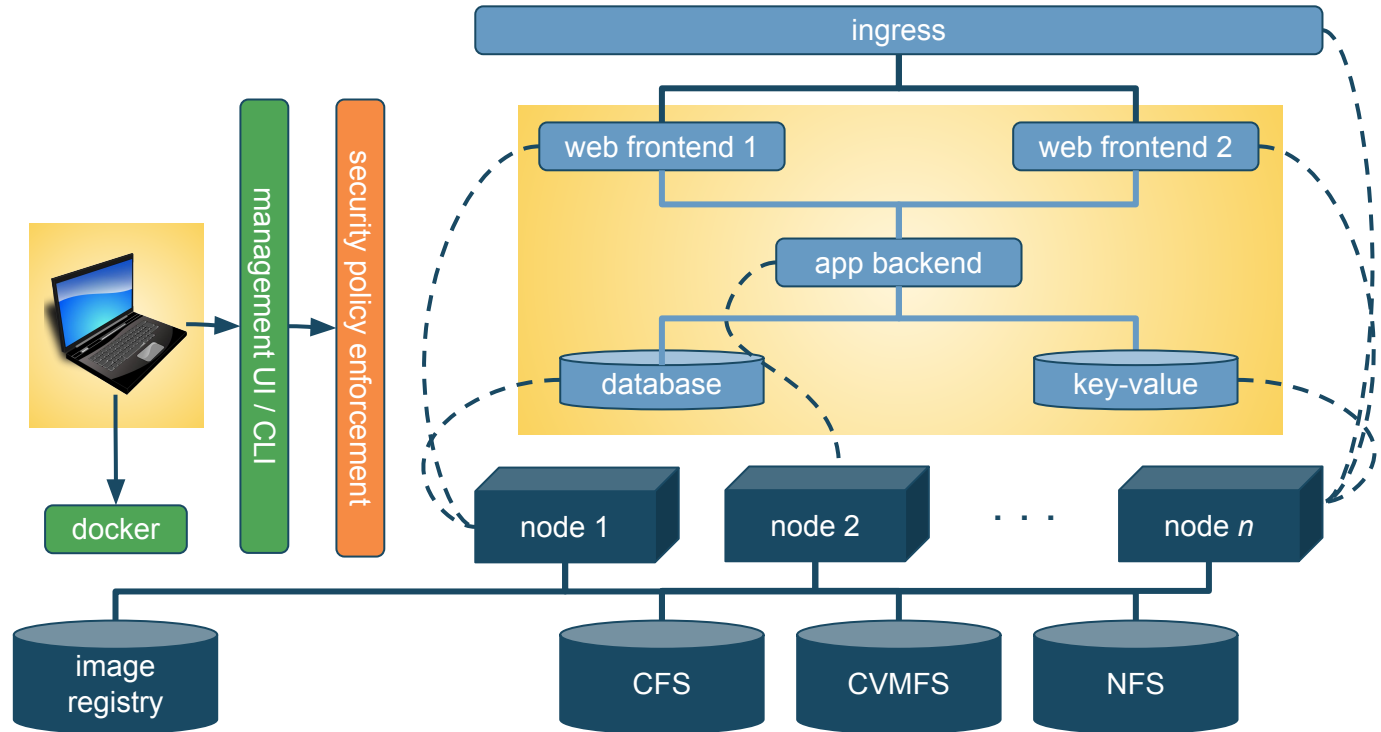


# Wrap-Up

# High-Level Spin Architecture

**Yours to  
manage**

**NERSC  
handles  
the rest!**



# Roles and Responsibilities

## You bring...

- **Your own microservice design**
- **Your own services based in Docker images**
- **Lifecycle management**
  - maintain at least one owner for every application
  - track Docker build files with git
  - minimize image customizations
- **Security management**
  - produce logs to stdout / stderr
  - use trustworthy public images; keep custom images updated
    - NERSC will scan images and network ports

# Roles and Responsibilities

## NERSC brings...

- **Stable infrastructure**
  - redundancy: 2x power, 2x network
  - dedicated storage
  - access to global file systems
- **Management practices for high uptime**
  - rolling upgrades
  - pre-scheduled quarterly maintenance
- **Support via the usual channels**
  - Spin team spans NERSC groups
  - NERSC staff are also Spin users!