SpinUp for New Users

https://www.nersc.gov/systems/spin/
Welcome

This workshop will prepare you to design, build, and manage your own apps using the Spin platform. Those might be:

- database-backed web apps that access project data
- workflow orchestration tools running outside of HPC
- API servers for real-time or distributed projects
- or something else!

Remember, though: Spin is for apps, not computation.
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!
Workshop Structure and Content

Seminar (today)
Learn concepts and terms. Build an example application. Store and access credentials. Configure storage and networking. Discuss the design and development process.

Hack-a-thon (choose A or B)
Try what you learned, in small groups, with hands-on help. Review. Q&A.

Ask questions here and on NERSC Users Slack (in #spin).
We welcome your feedback. Please complete our survey afterward.

Have a great workshop!
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 (97% in 2022)**
- **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 service that exposes a web service in a workload externally using a DNS name (layer 7)

**Load Balancer:** proxy service that exposes a non-web service in a workload using a DNS name (layer 2)

**Namespace:** group of workloads, ingresses, load balancers, volumes, config maps, secrets, etc

**Project:** group of namespaces; corresponds to a NERSC project

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

**Rancher:** *orchestration* system for Kubernetes clusters
Canonical Development Workflow

Build
images on your laptop with your custom software, and when they run reliably, ...

Ship
them to a registry for version control and safekeeping
- DockerHub: share with the public
- NERSC: keep private to your project

Run
your workloads

Spin

docker

HARBOR

NERSC

RANCHER

kubernetes
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
Try It Yourself!

1. Log in to [https://rancher2.spin.nersc.gov](https://rancher2.spin.nersc.gov).

2. In the sidebar under ≡, select development, then click Projects/Namespace.

3. Under your project (or if attending the SpinUp Workshop, under the spinup project), click Create Namespace. Enter a unique name, then click Create. Note: underscores (_) are not allowed!

4. In the sidebar under Workload, click Deployments and click Create. Select the namespace you just created and enter
   Name: `db`
   Container Image: `mysql:5`

5. Under Networking, click Add Port and enter
   Service Type: Cluster IP
   Name: `mysql`
   Private Container Port: 3306
   Protocol: TCP

6. Scroll down to Environment Variables, click Add Variable, and enter
   MYSQL_DATABASE = science
   MYSQL_USER = user
   MYSQL_PASSWORD = password1234
   MYSQL_RANDOM_ROOT_PASSWORD = yes
   TZ = US/Pacific

7. In the left panel, click Security Context and select
   Privilege Escalation: No
   Add Capabilities: CHOWN, DAC_OVERRIDE, FOWNER, SETGID, SETUID
   Drop Capabilities: ALL

8. Click Create.

9. Under the ⋮ menu to the right of your workload, select Execute Shell and enter
   # mysql -u user -D science -p
   (enter password from above)
   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.

<table>
<thead>
<tr>
<th>Capability</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHOWN</td>
<td>Change the owner of files and directories</td>
</tr>
<tr>
<td>DAC_OVERRIDE</td>
<td>Override file permissions</td>
</tr>
<tr>
<td>FOWNER</td>
<td>Override owner permissions</td>
</tr>
<tr>
<td>NET_BIND_SERVICE</td>
<td>Open network ports numbered &lt; 1024</td>
</tr>
<tr>
<td>SETGID</td>
<td>Change the group of a running process</td>
</tr>
<tr>
<td>SETUID</td>
<td>Change the user of a running process</td>
</tr>
</tbody>
</table>
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?
Try It Yourself!

1. In the sidebar, select **Storage > Secrets**. Click **Create**.
2. Select **Opaque**.
3. Set Values:
   - **Namespace**: Choose the namespace you created earlier.
   - **Name**: `db-password`
   - **Key**: `password`
   - **Value**: `<make-something-up>`
4. Click **Create**.

1. In the ⋮ menu for your workload, choose **Execute Shell** to look at the results:
   ```
   # cat /secrets/password
   ```
2. In the ⋮ menu, select **Edit Config**, expand **Environment Variables**, and replace `MYSQL_PASSWORD: password1234` with `MYSQL_PASSWORD_FILE: /secrets/password`
3. Click **Save**
4. To test the secret, click on your workload (db), select **Execute Shell**, and connect using the new password:
   ```
   # mysql -u user -D science -p
   ```
5. Notice: starting a new pod re-inited the database!
   ```
   mysql> show tables;
   Empty set (0.00 sec)
   ```
6. To view logs, click on name of the pod "db", and then select ⋮ > View Logs

---

**Use the secret**

---

**Attach the secret**
Discussion

- Secrets are a good way to manage and protect passwords, tokens, etc.
- 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)
### Deployments

<table>
<thead>
<tr>
<th>Name</th>
<th>Image</th>
<th>Ready</th>
<th>Upto-date</th>
<th>Available</th>
<th>Restarts</th>
<th>Age</th>
<th>Health</th>
</tr>
</thead>
<tbody>
<tr>
<td>app</td>
<td>registry.nersc.gov/spinui/galaxies</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>12 days</td>
<td></td>
</tr>
<tr>
<td>do</td>
<td>mysql5</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>12 days</td>
<td></td>
</tr>
<tr>
<td>jehf</td>
<td>ubuntu</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>12 days</td>
<td></td>
</tr>
<tr>
<td>mysql2</td>
<td>mysql5</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>19 days</td>
<td></td>
</tr>
<tr>
<td>mysql5</td>
<td>mysql5</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>26 days</td>
<td></td>
</tr>
<tr>
<td>mysql5</td>
<td>mysql5</td>
<td>1/1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>27 mins</td>
<td></td>
</tr>
</tbody>
</table>
Create and Mount the NFS Volume

1. In the sidebar under **Workload**, click **Deployments**, find your namespace, and click your **db** workload.

2. Under the ⋮ menu at the top right, click **Edit Config**; click **Pod**; in the left panel, click **Storage**.

3. Click **Add Volume**, select **Create Persistent Volume Claim**, and enter or select
   
   Persistent Volume Claim Name: *any name*
   
   Select **Use a Storage Class to provision…**
   
   Storage Class: **nfs-client**
   
   Access Modes: **Single-Node Read/Write**
   
   Capacity: **1 GiB**
   
   Volume Name: *any name, all lowercase*

4. Click **container-0**; in the left panel, click **Storage**.

5. Click "**Select Volume**" and choose the volume claim you just created.

6. Under **Mount Point**, enter `/var/lib/mysql`.

7. Leave **Sub Path in Volume** blank.

8. Click **Save**.

Test the Persistent Volume

1. Under 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. Under the ⋮ menu, click **Redeploy** and wait for the new container to be started.

3. Select **Execute Shell** again and check whether your changes persisted:

   ```
   # 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:
  o Flask to handle HTTP requests, routing, responses
    • Pretty simple galaxy cluster gallery app
  o Config map for setting some environment variable
  o Database for content and metadata
    • Stored on NFS
  o Image files for web front-end to serve up
    • Stored on CFS
Try It Yourself!

1. **Storage > ConfigMaps** then click “Create” then set:
   - Namespace: <your namespace>
   - Name: any name

2. **Set “Data” key/value pair:**
   - `banner_message = something hilarious`

3. **Click “Create” button**

4. **Workload > Deployments** then click “Create” then set:
   - Namespace: <your namespace>
   - Name: app
   - Under “container-0” tab in bottom middle set
     - **Container Image:** registry.nersc.gov/spinup/galaxies

5. **Scroll up** and click “Pod” tab and then click “Storage”
   - to configure two new volumes:
     - **Open** “Add Volume” dropdown, select “Bind-Mount” and set
       - **Volume Name:** vol-galaxydata
       - **Path on the Node:** /
global/cfs/cdirs/mpccc/rthomas/spin-demo/static
       - The Path on the Node must be: An existing directory

     - **Open** “Add Volume” dropdown, select “Secret” and set
       - **Volume Name:** vol-dbsecret, **Secret:** db-password

6. **Click “container-0” tab and then click “Storage”**
   - to attach new volumes to the Deployment
     - **Open** “Select Volume” dropdown, select “vol-galaxydata”, and set
       - **Mount Point:** /srv/static
       - **Read-Only:** [✔]

     - **Open** “Select Volume” dropdown, select “vol-dbsecret”, and set
       - **Mount Point:** /secrets
       - **Read-Only:** [✔]

7. **Click “container-0” tab** and then **click “Security Context”** and set
   - **Run as User ID:** <numeric user ID>
   - **Drop Capabilities:** ALL

8. **Click Pod” tab and then “Security Context”** and set
   - **Filesystem Group:** <numeric group ID>

9. **Click “Create” button**

---

To find your numeric user ID and a suitable numeric group ID, use the `id` command on a login node or go to Iris and check the Profile and Groups tabs.
Discussion: App, Behind-the-Scenes

• Where did the image come from?
  o Built image locally
  o [https://github.com/NERSC/spin-docker-compose-example](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
  o Push to registry.nersc.gov/<project>/<image-name>:<tag>

• How was the database initialized?
  o “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

<table>
<thead>
<tr>
<th>Storage Type</th>
<th>Persistent</th>
<th>On HPC</th>
<th>Size</th>
<th>Best Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Global File Systems</strong> <strong>(CFS, Common)</strong></td>
<td>✓</td>
<td>✓</td>
<td>O(quota)</td>
<td>sequential</td>
</tr>
<tr>
<td><strong>NFS</strong></td>
<td>✓</td>
<td></td>
<td>O(10GB)+</td>
<td>random</td>
</tr>
<tr>
<td><strong>CVMFS (read-only)</strong> <strong>always mount at root!</strong></td>
<td>✓</td>
<td>✓</td>
<td>n/a</td>
<td>CERN software</td>
</tr>
<tr>
<td><strong>in-container</strong></td>
<td></td>
<td></td>
<td>O(1GB)</td>
<td>temporary</td>
</tr>
</tbody>
</table>
## Discussion: Storage Options

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

*What other examples? What are some exceptions?*
Exercise 5: Networking
Exercise 5: Networking (Internal / Overlay)

Traffic between containers uses a **private overlay network**.

- Each container gets an IP within 10.42.*.*
- IPs change when new containers are created!
- DNS names are automatically created (and updated)
  
  `<workload> [.<namespace> [.svc.cluster.local]]
  
  => 10.42.x.y

For example, the database in our example app:

  `db.<namespace>.svc.cluster.local`, or simply
  
  `db`
Exercise 5: Networking (External Inbound)

HTTP traffic requires an *Ingress*.

- When you create an ingress, a dynamic DNS name is associated with it; the workload(s) you specify become accessible on port 80.
  
  `<ingress name>.<namespace>.<cluster>.svc.spin.nersc.org`  
  `=> ingress controller IP address(es)`  

- You must add a friendly name and matching web certificate.
- Redirection to HTTPS happens automatically.
- Many aspects can be configured with *annotations*. 
Exercise 5: Networking (External Inbound)

Non-HTTP traffic requires a Load Balancer.

• A dynamic DNS name is associated with the workload; it becomes accessible at the port you specify
  
  `<workload>-loadbalancer.<namespace>..<cluster>.svc.spin.nersc.org`  
  `=> 128.55.212.*` (dedicated IP for this load balancer)

• Only accessible from NERSC networks.

• Common ports are allowed; let us know if you need others:
  
  3306, 4873, 5432, 5672, 5984, 15672, 27017

Outbound external traffic just works (via NAT).
### Deployments

<table>
<thead>
<tr>
<th>Namespace: jfroula-mysql</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>app</strong></td>
</tr>
<tr>
<td><strong>db</strong></td>
</tr>
<tr>
<td><strong>jettapp</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Namespace: mydb2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>db</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Namespace: shalapp</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>db</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Namespace: unique-newyork</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>app</strong></td>
</tr>
<tr>
<td><strong>db</strong></td>
</tr>
</tbody>
</table>
Try It Yourself!

1. **Workload > Deployments** in left navigation menu
2. **Click** the ⋮ menu to the right of your “app” workload, and **select** “Edit Config”
3. In “container-0” tab, **scroll down**, **click** “Add Port” button and **set**
   - Service Type: Cluster IP
   - Private Container Port: 5000
   - Name: flask
   - Protocol: TCP

4. **Click** “Save” button in the lower left to redeploy “app”.

1. **Service Discovery > Ingresses** in left navigation menu
2. **Click** the “Create” button in the upper right
3. **Set** these values
   - Namespace: <Namespace from previous exercise>
   - Name: ingress
   - Request Host: ingress.<namespace>.development.svc.spin.nersc.org
   - Path: /  
     **Watch out for leading or trailing spaces**
   - Target Service: app
   - Port: 5000
4. **Click** “Create” button in the lower left.

You are back at the **Service Discovery > Ingresses** screen
1. **Wait** for state tag to change to **Active**
2. **Wait** for DNS to propagate to the LBL/NERSC and other DNS servers (Usually 1-5 minutes)
3. **Access** your app at: [http://ingress.<namespace>.development.svc.spin.nersc.org](http://ingress.<namespace>.development.svc.spin.nersc.org)
Exercise 5: Add a Friendly Hostname

Example: www.cosmosgallery.org

1. Request a DNS CNAME record in this format from your DNS provider:
   <friendly name> CNAME
   <ingress>.<namespace>.<development or production>.svc.spin.nersc.org.

   For example,
   This will typically take a day or more.

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
Try It Yourself Later: Add a Friendly Name and SSL

1. Get a CNAME entry from your DNS provider that points at your ingress. For instance:
   
   `<friendly name> CNAME lb.<namespace>.<cluster>.svc.spin.nersc.org`

2. When it is ready (hours or days later), navigate to **Service Discovery > Ingresses** in Rancher.

3. **Click** the ⬤ icon next to your ingress, and **select** “Edit Config” from the dropdown.

4. **Click** Add Rule.

5. For **Request Host** enter the friendly name. Do not alter the existing rule.

6. Select the same **Path**, **Target Service** and **Port** as in the existing ingress rule, then click **Save**.

---

SSL/TLS (HTTPS)

1. Get a web certificate from your provider. There are many tutorials on how to do this.

2. Navigate to **Storage > Secrets**, click the **Create** button, then click **TLS Certificate**.

3. Enter a meaningful **Name** and select a **Namespace** scope.

4. Upload your **Private Key** and **Certificate** using “Read from File” buttons and click **Create**.

5. Navigate to **Service Discovery > Ingresses**.

6. **Edit** the ingress, select **Certificates** panel, click **Add Certificate**, select your certificate from the list, add your hostname, and **Save**.

---

Friendly Hostname
Try It Yourself Later: Add a Load Balancer

1. Under Workload, click Deployments and click your db workload; under the menu, click Edit Config; in the header, click container-0; in the left panel, click General. Scroll down to Ports.

2. Modify the existing port
   - Service Type: Load Balancer
   - Name: mysql
   - Private Container Port: 3306
   - Protocol: TCP
   - Listening Port: 3306

   Only common ports are exposed!

   (Don’t pick your “favorite” port here)

3. Click Save.

4. Under Service Discovery, click Services and find your namespace; your load balancer will be named db-loadbalancer.

5. Just like an ingress, Rancher will create a DNS name for the load balancer based on the workload name, namespace, and cluster name. To see this generated name, click db-loadbalancer, then Show # annotations near the top of the page.

6. To try connecting to the database via the load balancer, log into Perlmutter and run

   ```bash
   % mysql -p -u user -D science -h db-loadbalancer.namespace.development.svc.spin.nersc.org
   ```
Discussion: Networking

• Beware of DNS propagation delays
  o Wait a minute for the DNS name of a *brand new* ingress or load balancer to be created; Rancher uses an internal queue.
  o Accessing either too early can negative cache for five minutes.

• Custom hostnames and web certificates
  o Processes vary for obtaining a hostname and certificate.
  o Check with your institution or PI.

• Web certificate chain ordering
  o If your certificate requires a chain, use nginx ordering: your certificate first, then that of its issuer, etc, but omitting the root.
Viewing Logs and Performance Data
### Viewing Logs, Events & Conditions

<table>
<thead>
<tr>
<th>Log Type</th>
<th>Content</th>
<th>Where</th>
<th>Best Use</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Container</strong></td>
<td>All stdout and stderr from container processes</td>
<td><strong>Option #1:</strong> <em>Workload &gt; Deployments</em>, click on the name to view the pods, select <em>View Logs</em> under (⋮) menu next to pod</td>
<td>Application problem, but container runs</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Option #2:</strong> <em>Workload &gt; Pods</em>, select <em>View Logs</em> under (⋮) menu next to pod</td>
<td>Container produces error at startup, exits, and restarts</td>
</tr>
</tbody>
</table>
| **Workload Events, Pod Conditions** | Scheduler activity (start, stop, scale), node failure                  | *Workload > Deployments* > *(Your Deployment)* and choose one of:  
  - *Recent Events*  
  - *Conditions*                                                 | Workload will not start or scale at all                                                        |
|                 |                                                                         |                                                                                                                                         | Container restarts continuously                                                               |
|                 |                                                                         |                                                                                                                                         | Denied due to security policy                                                                 |

---

**Option #1:** *Workload > Deployments*, click on the name to view the pods, select *View Logs* under (⋮) menu next to pod.

**Option #2:** *Workload > Pods*, select *View Logs* under (⋮) menu next to pod.
Performance Analytics

Rancher provides live Grafana plots of Kubernetes Resource Metrics:

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

Where:

- **Workload > Deployment**, click **Metrics** tab
Building Your Own Microservices App
Microservices

Services (Workloads/Pods)
- Valuable actions that fulfill a demand
- One or more containers

Microservice Architecture
- How services are combined

Service Principles
- Modular and loosely coupled
- Composable
- Platform and language independent
- Self-describing
Starting Your Microservice Design

Why should you think about your app in terms of microservices?

What are some examples of microservice components?

What does Spin take care of or make easy for you?

*Draw a microservices picture of your use case!*

How does your app get on the web?

What conventions do we recommend?
Categories of Microservices

Web Front-end
Web App • Authentication • Access Control

Application Logic
REST API • Workflow Engine

Metadata, Application State, or Science Data
SQL • NoSQL • XML

File Storage for Science Data
Ephemeral or Persistent • Open or Closed

What are some others?
Real-World Example: jupyter-test

PostgreSQL database

NFS (database tables)

(Not the sqlite default for app state.)

Persistent, not exposed to users or other systems.

JupyterHub web front-end

NFS (certs)

Reverse Proxy Ingress

Notebook Node API server

Global File System

SSH Service API server

NFS (database tables)

NFS (certs)

Persistent and exposed to users and other systems.

Reverse Proxy

Ingress

JupyterHub

Node

API server

Global File System

Persistent and exposed to users and other systems.
Recall: Container Development Workflow

Build
images on your laptop with your custom software, and when they run reliably, ...

Ship
them to a registry for version control and safekeeping
- DockerHub: share with the public
- NERSC: keep private to your project

Run
your workloads

Spin

NERSC
‘Classic’ Development Model

1. Build your app on your laptop like in the big picture.

2. Run and test containers locally.

3. Use mock APIs or mock volume mounts with a subset of data on your laptop.

4. For the brave: mount larger data sets over sshfs, but...
“Classic” Development Model

Pros:

● Testing on your laptop is a tight loop.
● No deployment to Spin until things are working reliably.

Cons:

● Pushing big images with small bandwidth is slow.
● Complex apps can be difficult to build in a simple local setup.
“On Ramp” Development Model

(This applies mostly to web front-ends.)

1. Docker image houses your application dependencies and runs with your UID and GID.

2. Deploy “app.py” code to some path on CFS with appropriate permissions.

3. Mount app.py’s directory and run it with “reload on source change” turned on.

4. Now you can hack in traditional fashion. (Eventually move “app.py” into container.)
"On Ramp" Development Model

**Pros:**
- Less pushing images from your laptop.
- No setting up of mocks APIs or mounts/sshfs.

**Cons:**
- You depend on a “data” filesystem for hosting code.
- Tendency to build up technical debt especially as new deps arise.
“DevOps” Model is Ideal

1. Starts out like classic model.
2. Leverage continuous integration to automate image build, test, and push to registry.
3. Trigger re-deployment on successful image push and test.
4. Not all features available yet.
“DevOps” Model is Ideal

Pros:

● Most efficient and reliable.
● Promotes inner peace.
● Keep computers busy; delay the singularity. (ahem)

Cons:

● Requires setup and commitment from team.
● Not all the tools available (yet) in Spin.
Encouragements and Admonishments

We most extremely strongly admonish you not to use docker commit. It enables changes that go untracked and are not easily reproduced. Changes to your Dockerfile should be under source control. It should feel wrong to you.

Iterating a lot on an image build?
To force rebuilds from a point just insert `RUN env` or `RUN pwd` to force the build from that point (c.f. multi-stage builds).

Want to start all over with a clean slate?
Use the `--no-cache` option in your docker build.

Need to clean out containers and images?
- `docker rm -f $(docker ps -aq)`
- `docker rmi -f $(docker images -q)`
- `docker container prune`
- `docker image prune`
- `docker system prune`
Wrap-Up
High-Level Spin Architecture

Yours to manage

NERSC handles the rest!

Your image registry

Management UI/CLI

Security policy enforcement

Ingress

Web frontend 1

Web frontend 2

App backend

Database

Key-value

Node 1

Node 2

... node n

CFS

CVMFS

NFS

Your Registry

Your UI/CLI

Your Security Policy Enforcement

Your Ingress

Your Web Frontends

Your App Backend

Your Database

Your Key-value Storage

Your Nodes

Your Container Images
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!
Questions and Hack-a-thon Prep