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 (99% in 2020 and 2021)
- 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.
(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 (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

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.
  o Frequently updated, easy to customize...less work!
• Look at the README: https://hub.docker.com/_/mysql
  o Customize by setting variables; no custom image needed
Watch an Example

2. Under Global, select the development cluster, then select a project. (If attending the SpinUp Workshop, use the spinup 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, expand Security & Host Config, and under Drop Capabilities, select ALL

7. Under Add Capabilities, select
   
   CHOWN, DAC_OVERRIDE, FOWNER, SETGID, SETUID

8. Click Launch and watch the pod start up.

9. 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
  o Read the READMEs for how to set variables
  o 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?
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

7. Use Exec Shell to look at the results
   # cat /secrets/password

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

9. Click Save

   Use the secret

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

11. Use Exec Shell again and use the new password to connect to MySQL
    # mysql -u user -D science -p

12. Notice: starting a new pod re-initiated the database!
    mysql> show tables;}
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:
  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
Watch an Example
Try It Yourself!

1. **Resources > Workloads** then **click “Deploy”**
   - **Namespace**: `<your namespace>`
   - **Docker Image**: `registry.nersc.gov/spinup/galaxies`

2. **Set** “Config Map Values” key/value pair:
   - `banner_message = <something hilarious>`

3. **Click** “Save” button

4. **Click** “Show advanced options” to see more panels, **expand “Command” panel**, and **set**
   - **User ID**: `<user ID>`
   - **Filesystem Group**: `<group ID>`

5. **Expand** “Security & Host Config” panel and **set**
   - **Drop Capabilities**: ALL

6. **Click** “Launch” button

---

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

• 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 (Homes, CFS)</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></td>
<td>✓</td>
<td>✓</td>
<td>n/a</td>
<td>CERN software</td>
</tr>
<tr>
<td><strong>always mount at root!</strong></td>
<td></td>
<td></td>
<td></td>
<td></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).
Watch an Example: Create an Ingress

Add Ingress

Name

Add a Description

Namespace

Add to a new namespace

Rules

- Automatically generate a hostname
- Specify a hostname to use
- Use as the default backend

Request Host

Target Backend
Try It Yourself!

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

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

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
Watch an Example: Add a Friendly Hostname
Watch an Example: Add a Web Certificate
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 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.

---

Try It Yourself! (Later; For Future Reference)

1. Get a web 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.
1. Edit the database workload (the one that you made in exercise 1). This is different from the HTTP ingress which is edited from the “Load balancing” tab.

2. Under port mapping enter:

   - Port Name: mysql
   - Publish the container port: 3306
   - Protocol: TCP
   - As a Layer-4 Load Balancer
   - On listening port: 3306

   Only common ports are exposed!
   (Don't pick your “favorite” port here)

3. Click Save. A load balancer should have automatically appeared in the “load balancing” tab, and the database should be accessible at NERSC (on Spin, Cori, and Perlmutter) under:

   `<workload>-loadbalancer.<namespace>.<cluster>.svc.spin.nersc.org`

---

**Try It Yourself! (Later: For Future Reference)**

1. Log into Cori
2. Run:

   `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

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

NFS (certs)

SSH Service
API server

Global File System

Persistent and exposed to users and other systems.

Persistent, not exposed to users or other systems.

(Not the sqlite default for app state.)
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
“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.
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!

image registry

docker

management UI/CLI

security policy enforcement

node 1

node 2

node n

CFS

CVMFS

NFS

database

key-value

app backend

web frontend 1

web frontend 2

ingress

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!
Questions and Hack-a-thon Prep