HPC GitOps Automation

August 7, 2024

Nikolas Rodriguez and Twinkle Wilson

Mentors: David Fox and Martin Baltezore

Team Members!

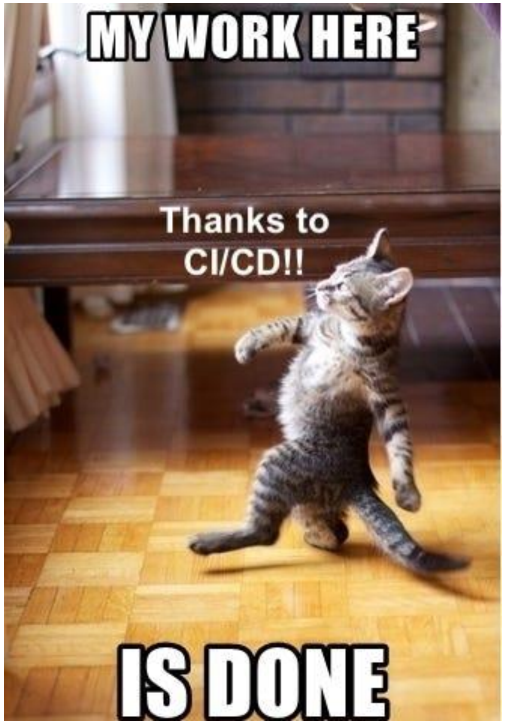
- Twinkle Wilson
- California State University East Bay
- Computer Science
- Expected Grad: May 2026

- Nikolas Rodriguez
- University of California, Los Angeles
- Computer Science
- Expected Grad: May 2025

Our Goals!

- Install GitLab runners for the HPC Academy
- Configure the runners to automatically start and stop based on demand — using Kubernetes
- Automate full cluster installation with one button push in GitLab.
- In a Gitlab repository, have code that does the following:
  - **Boot Up a Management Node**
  - **Configure Management Node**
  - **Boot Up Compute Nodes**
  - **Configure Key Services**
  - **Run a Slurm job to verify the entire setup**

CI/CD kubernetes



So… how did we get started?

- **Initial Setup:**
  - Installed GitLab on compute nodes
  - Connected to Firefox from the management node via VNC Viewer
  - Installed and registered runner on a separate node, initially using shell executor

- **The Challenge: Inefficient Resource Utilization by Runner**
  - Our runner was constantly polling for new jobs even when there were none.

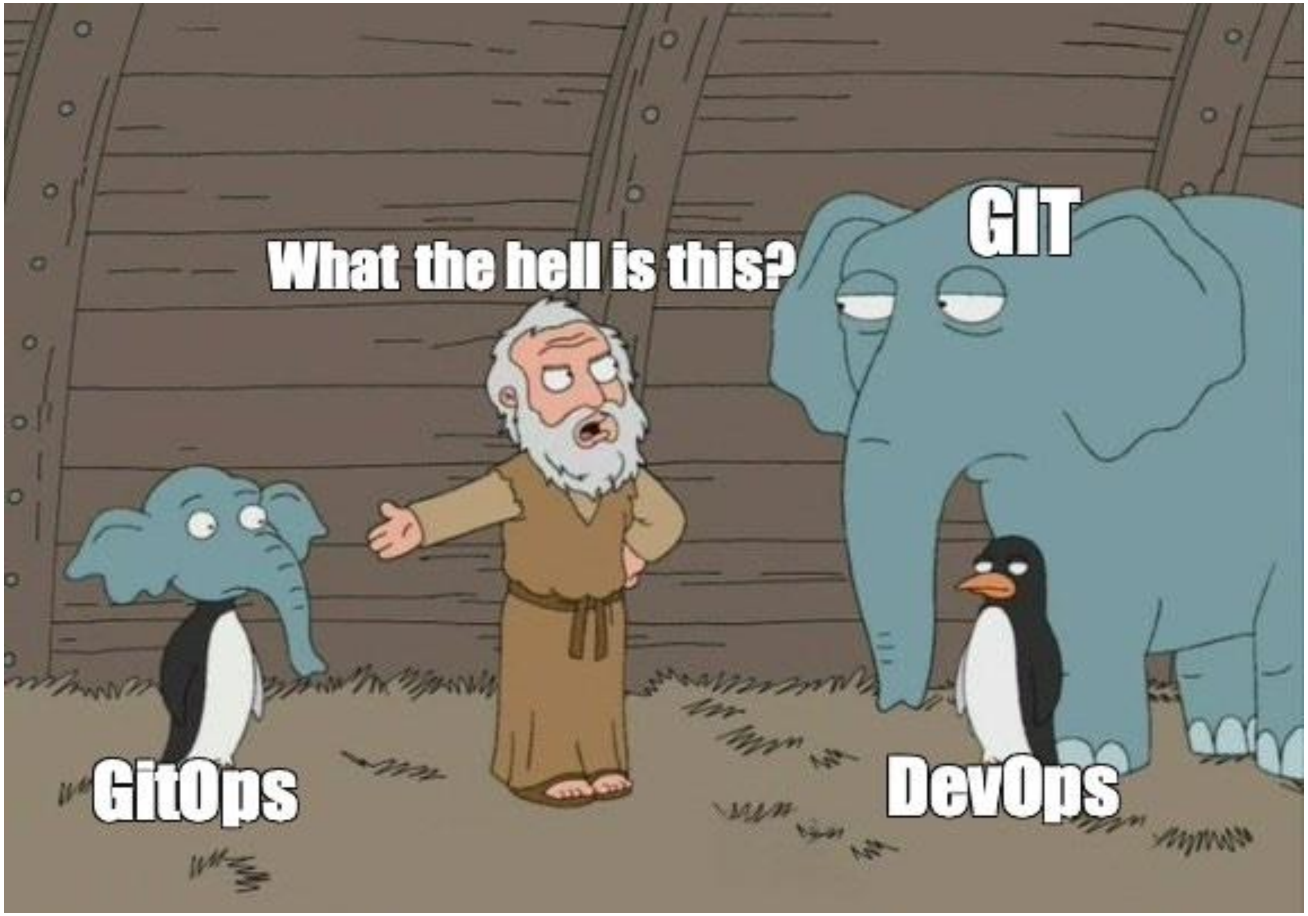
- **The Solution: Deploying Runner in Kubernetes**
  - We switched to using Kubernetes to manage our runner. Now, instead of every runner always polling for jobs, Kubernetes creates a new pod for each job with one manager runner.

Different Types of Executors

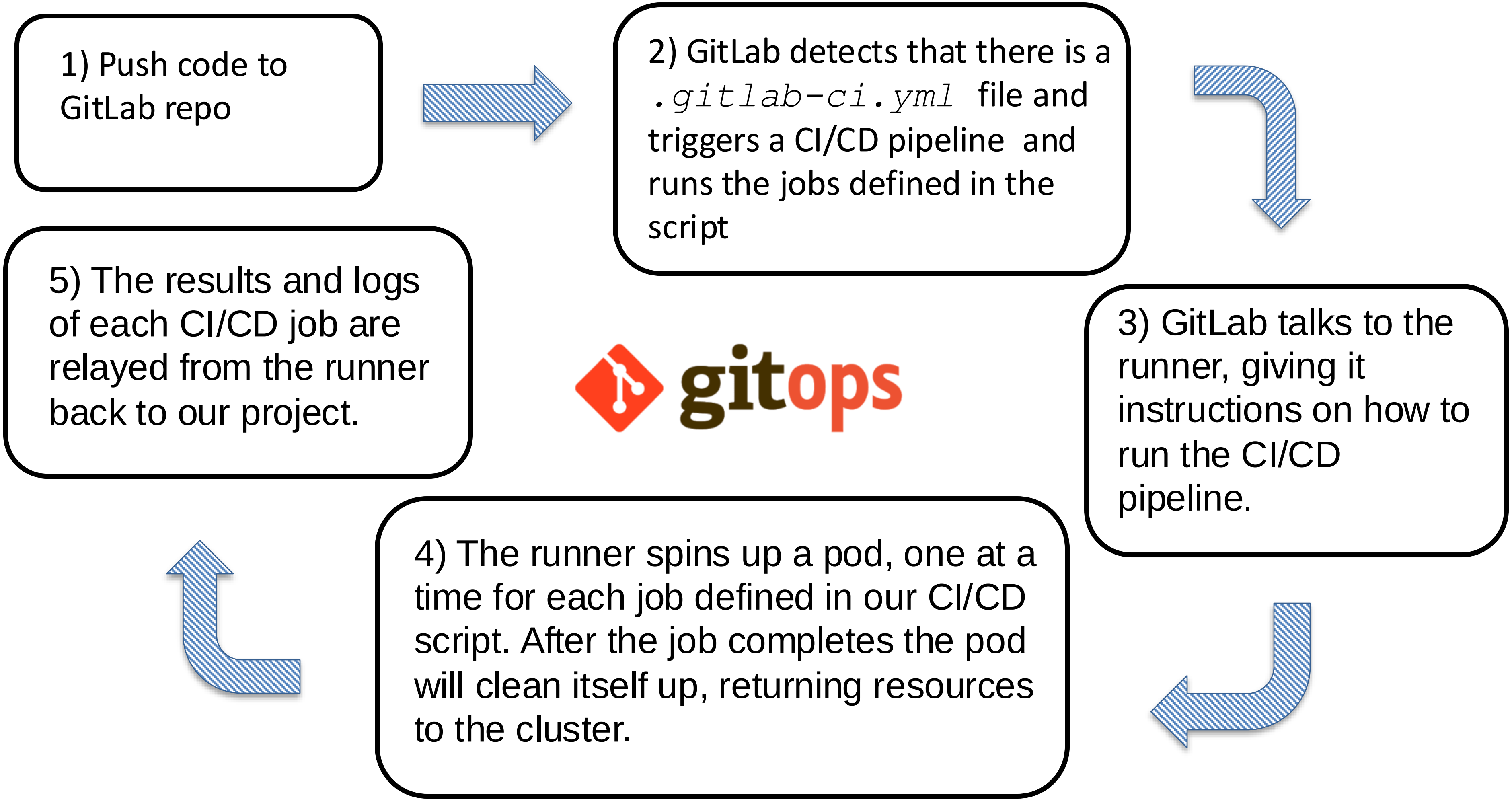
Each GitLab Runner defines at least one executor, which is the environment in which the job(s) will be executed. Within GitLab, you can use different executors, depending on your needs:

- **SSH:** Run jobs on remote machines via SSH.
  - For remote servers.
- **Shell**: Execute jobs directly in the shell.
  - For simplicity.
- **Docker**: Use Docker containers for your jobs.
  - For isolated, reproducible environments.
- **Kubernetes**: Run jobs in Kubernetes pods.
  - For cloud-native scaling.
- **Custom**: Create your own custom executor.
  - For unique requirements.
- **VirtualBox:** Run jobs in VirtualBox VMs.
  - For full VM isolation.

What's GitOps, anyway?



GitOps Workflow Diagram!



1) Push code to GitLab repo

2) GitLab detects that there is a `.gitlab-ci.yml` file and triggers a CI/CD pipeline and runs the jobs defined in the script

3) GitLab talks to the runner, giving it instructions on how to run the CI/CD pipeline.

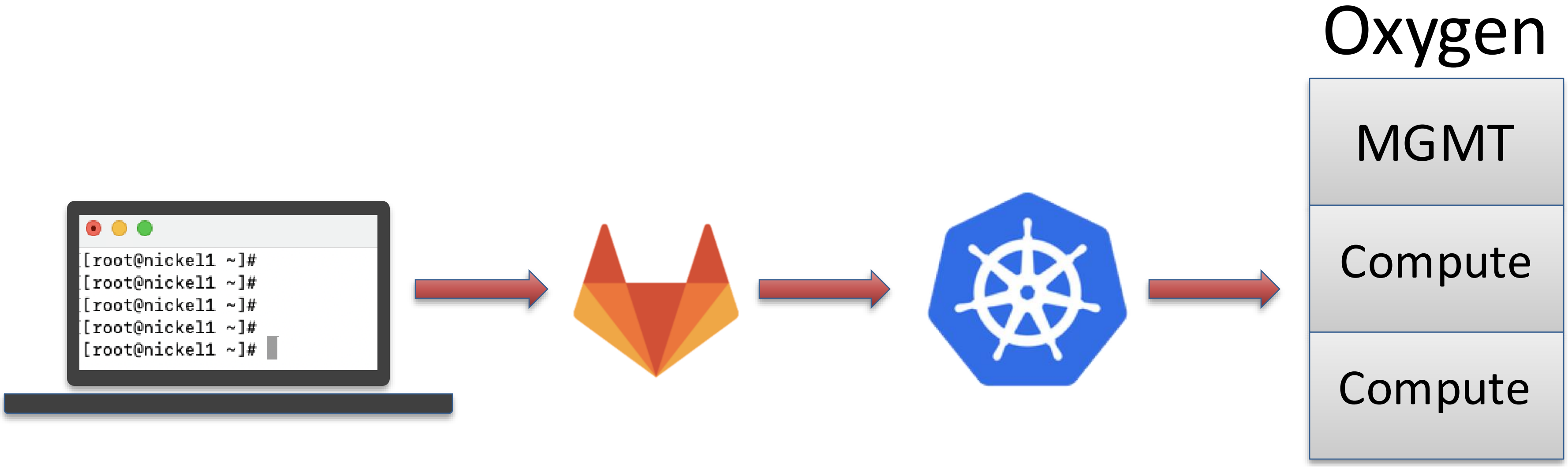
4) The runner spins up a pod, one at a time for each job defined in our CI/CD script. After the job completes the pod will clean itself up, returning resources to the cluster.

5) The results and logs of each CI/CD job are relayed from the runner back to our project.

Our Specialized Runner – Cluster Build Automation

- How could we
  - Demonstrate a GitOps workflow + the power of runners
  - Stay within the spirit of the HPC Cluster Academy

- Solution: Create a runner to fully automate a cluster build.
  - Ideally, in under an hour (!).



```
[root@nickel1 ~]#
[root@nickel1 ~]#
[root@nickel1 ~]#
[root@nickel1 ~]#
[root@nickel1 ~]#
```

Oxygen

MGMT

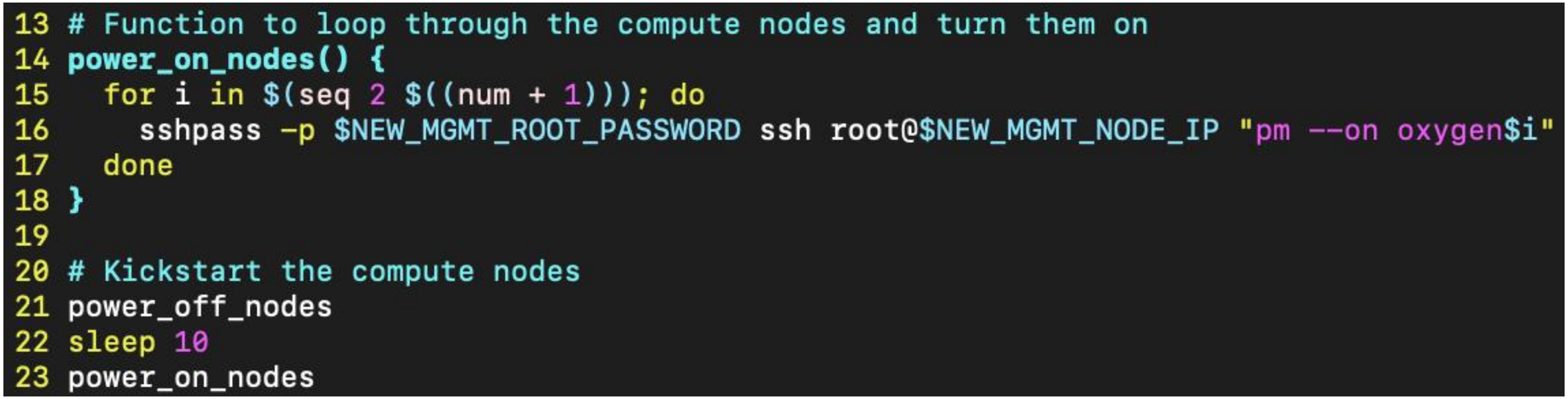
Compute

Compute

Early Stages of Our Runner!

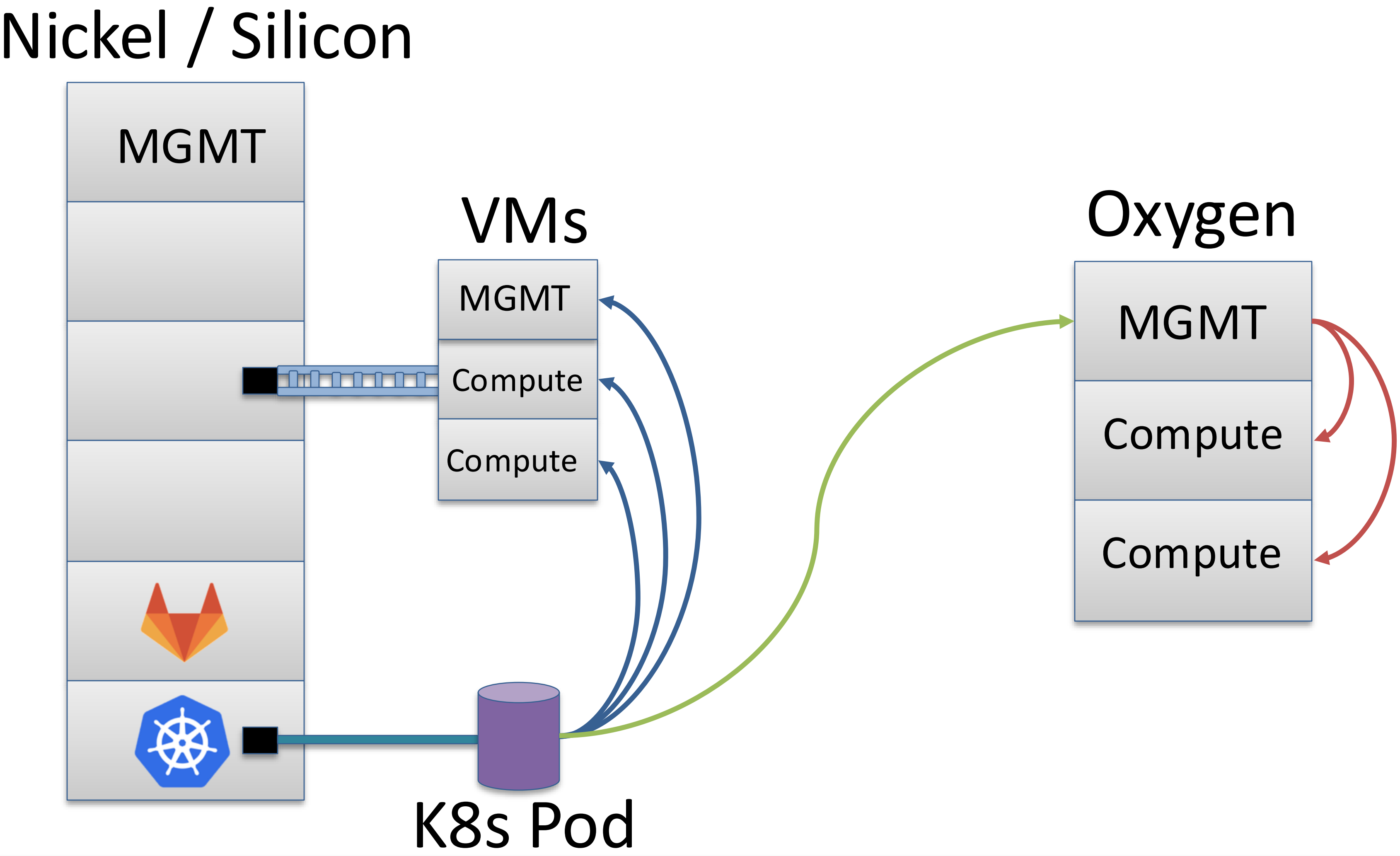
- We started by using our runner and simple ansible playbooks to "configure" VMs just to get the concept down.

- Later began working with physical compute nodes (Oxygen cluster).
  - Assumed we had a configured management node.
  - Had to adjust to fundamental difference between physical and virtual nodes.



```
# Function to loop through the compute nodes and turn them on
power_on_nodes() {
  for i in $(seq 2 $((num + 1))); do
    sshpass -p $NEW_MGMT_ROOT_PASSWORD ssh root@$NEW_MGMT_NODE_IP "pm --on oxygen$i"
  done
}

# Kickstart the compute nodes
power_off_nodes
sleep 10
power_on_nodes
```

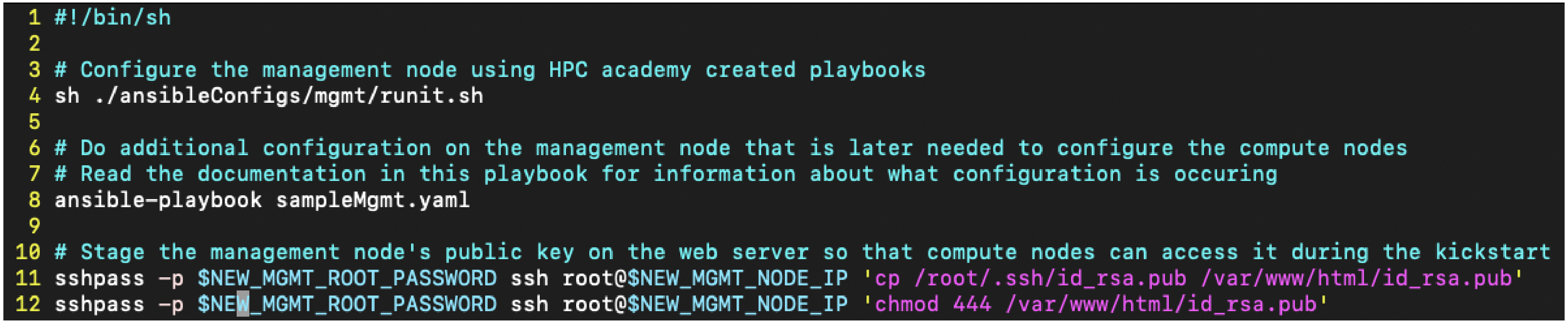
Configuring VMs vs. Configuring Physical Nodes!



Kickstarting a Management Node!

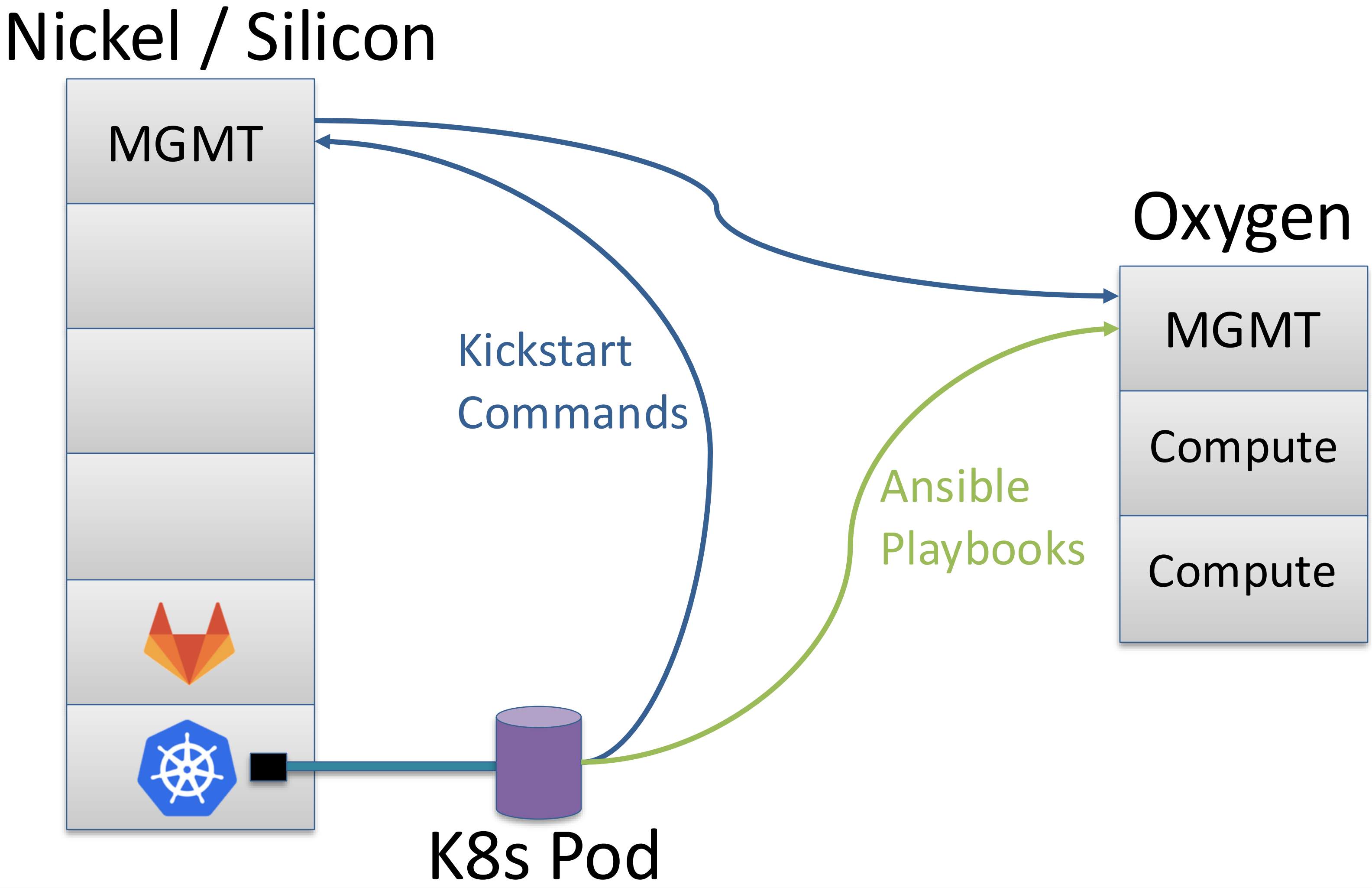
- Something we hadn’t done before.
    - Although closely related to kickstarting a compute node.

- Once kickstarted installed, configured using academy written ansible playbooks.
    - Also included some project specific configuration.

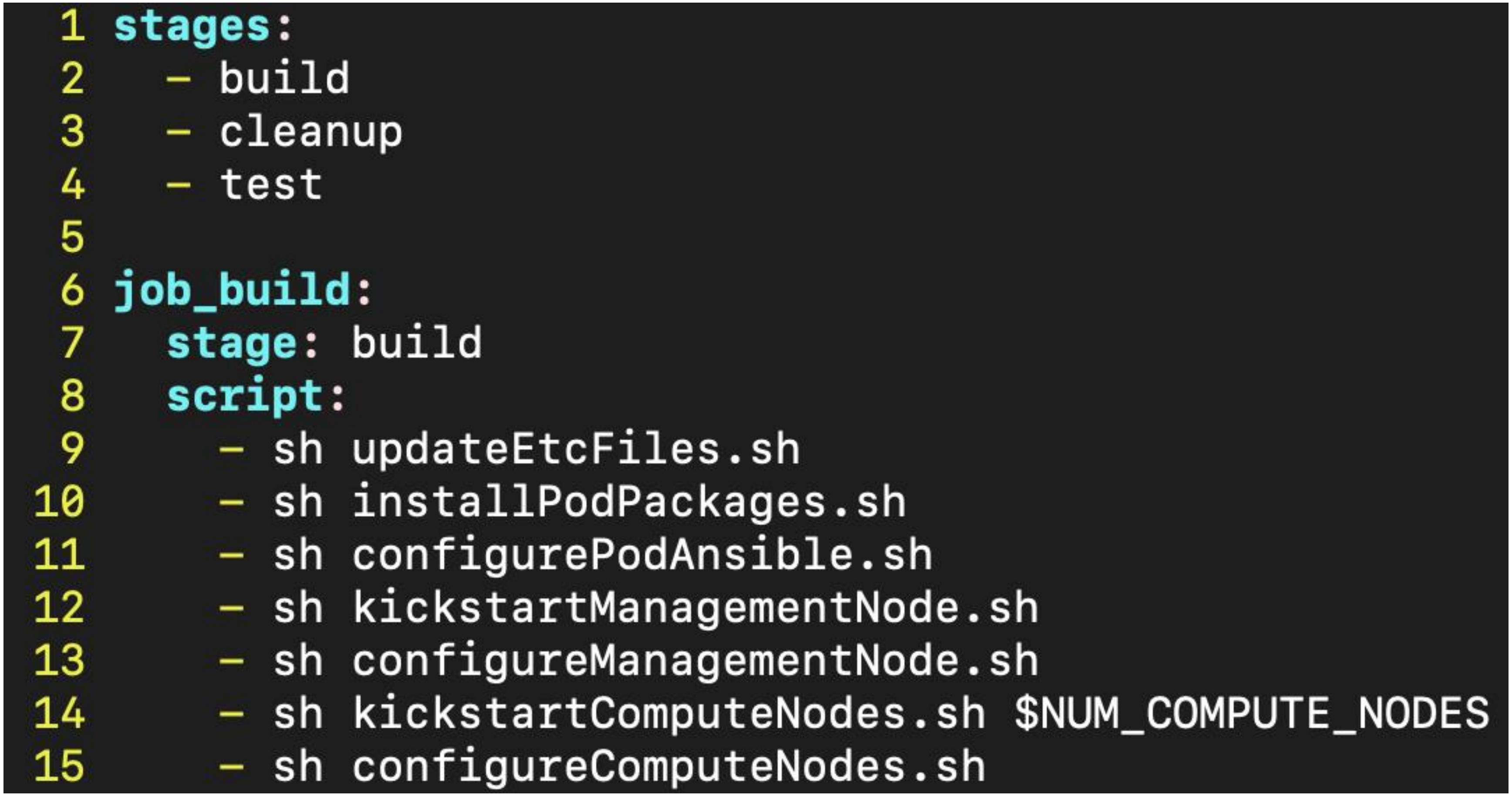


```
#!/bin/sh

# Configure the management node using HPC academy created playbooks
sh ./ansibleConfigs/mgmt/runit.sh

# Do additional configuration on the management node that is later needed to configure the compute nodes
# Read the documentation in this playbook for information about what configuration is occuring
ansible-playbook sampleMgmt.yaml

# Stage the management node's public key on the web server so that compute nodes can access it during the kickstart
sshpass -p $NEW_MGMT_ROOT_PASSWORD ssh root@$NEW_MGMT_NODE_IP 'cp /root/.ssh/id_rsa.pub /var/www/html/id_rsa.pub'
sshpass -p $NEW_MGMT_ROOT_PASSWORD ssh root@$NEW_MGMT_NODE_IP 'chmod 444 /var/www/html/id_rsa.pub'
```

Kickstarting and Configuring a Management Node!

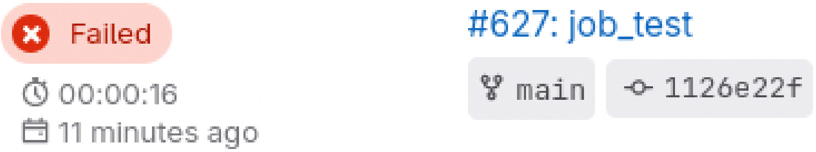


Putting Everything Together!



```
stages:
  - build
  - cleanup
  - test

job_build:
  stage: build
  script:
    - sh updateEtcFiles.sh
    - sh installPodPackages.sh
    - sh configurePodAnsible.sh
    - sh kickstartManagementNode.sh
    - sh configureManagementNode.sh
    - sh kickstartComputeNodes.sh $NUM_COMPUTE_NODES
    - sh configureComputeNodes.sh
```

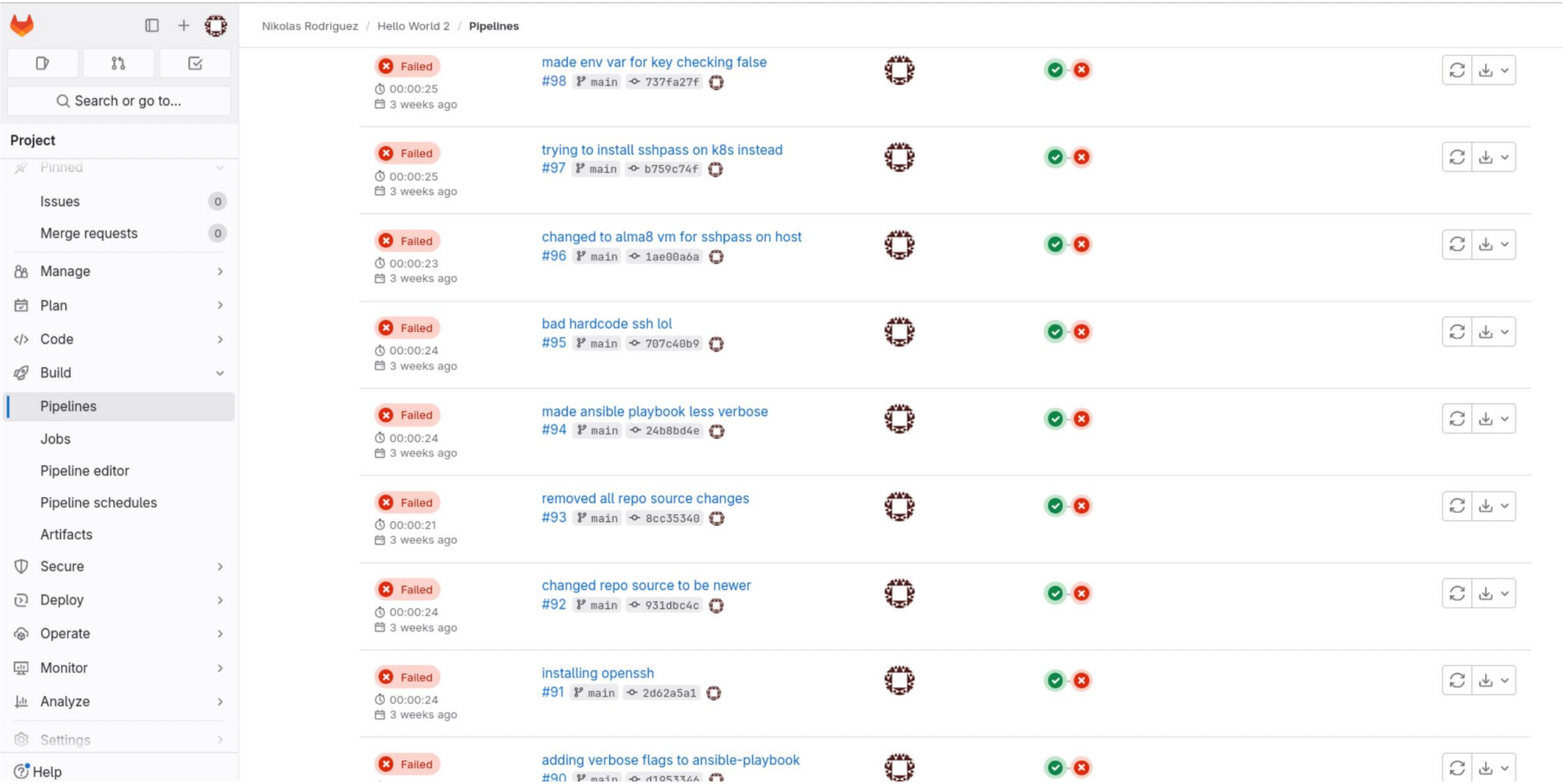
Challenges



00:00:16
11 minutes ago

#627: job_test
main
1126e22f

T



Nikolas Rodriguez / Hello World 2 / Pipelines

Failed 00:00:25 3 weeks ago — made env var for key checking false #98 main 737fa27f

Failed 00:00:25 3 weeks ago — trying to install sshpass on k8s instead #97 main b759c74f

Failed 00:00:23 3 weeks ago — changed to alma8 vm for sshpass on host #96 main 1ae00a6a

Failed 00:00:24 3 weeks ago — bad hardcode ssh lol #95 main 707c40b9

Failed 00:00:24 3 weeks ago — made ansible playbook less verbose #94 main 24b8bd4e

Failed 00:00:21 3 weeks ago — removed all repo source changes #93 main 8cc35340

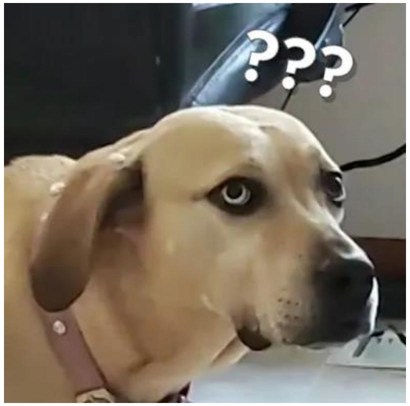
Failed 00:00:24 3 weeks ago — changed repo source to be newer #92 main 931dbc4c

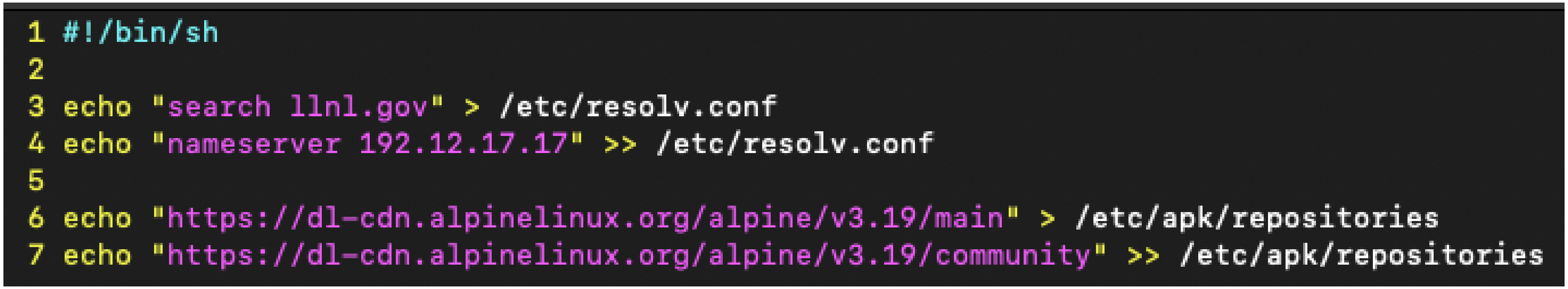
Failed 00:00:24 3 weeks ago — installing openssh #91 main 2d62a5a1

Failed — adding verbose flags to ansible-playbook #90 main d1953346

#1: Obtaining Pod Dependencies

- Images offered by Gitlab were extremely minimal
  - Needed to install dependencies ourselves.
- Experienced DNS issues.
- Container's packages were initially incompatible with our OS install.



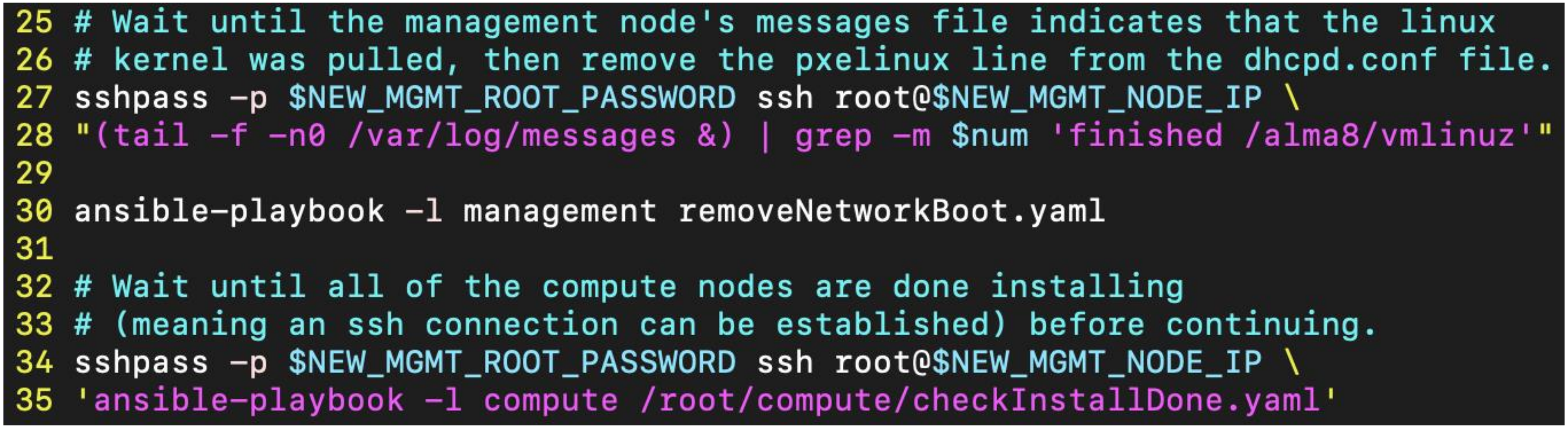


```sh
#!/bin/sh

echo "search llnl.gov" > /etc/resolv.conf
echo "nameserver 192.12.17.17" >> /etc/resolv.conf

echo "https://dl-cdn.alpinelinux.org/alpine/v3.19/main" > /etc/apk/repositories
echo "https://dl-cdn.alpinelinux.org/alpine/v3.19/community" >> /etc/apk/repositories
```

#2: Detecting Status of OS Install

- Needed to detect when the OS install had started.
    - Otherwise the node would infinitely network boot.

- Needed to detect when the OS install was complete.
    - Otherwise the runner would try (and fail) to configure the node with ansible.



```
# Wait until the management node's messages file indicates that the linux
# kernel was pulled, then remove the pxelinux line from the dhcpd.conf file.
sshpass -p $NEW_MGMT_ROOT_PASSWORD ssh root@$NEW_MGMT_NODE_IP \
"(tail -f -n0 /var/log/messages &) | grep -m $num 'finished /alma8/vmlinuz'"

ansible-playbook -l management removeNetworkBoot.yaml

# Wait until all of the compute nodes are done installing
# (meaning an ssh connection can be established) before continuing.
sshpass -p $NEW_MGMT_ROOT_PASSWORD ssh root@$NEW_MGMT_NODE_IP \
'ansible-playbook -l compute /root/compute/checkInstallDone.yaml'
```

If we had more time, we would…

- Include runner dependencies in a custom image.
  - Could be in a VM image using libvirt executor.
  - Could be in a container image (if / when supported by Gitlab).

- Optimize academy written ansible playbooks for scalability.
  - Current playbooks assume 2 compute nodes.
    - Would want to work with ansible loops instead of hardcoding.
  - Already started this scalability mission within our runner’s script.

Thank you to…

Our mentors!

David Fox

Martin Baltezore

Special Thanks to…

Everyone Else Who Contributed!

- Alicia Gullon
- Jeremi Nuer
- Kendall Parry
- Lucian Chauvin
- Gabe Maxfield
- Naomi Cheeves
- Jason Kim
- John Consolati

Sources!

- https://docs.gitlab.com/ee/ci/pipelines/
- https://youtu.be/mnYbOrj-hLY?si=MqlLca0Nsrpg2r7x
- https://docs.gitlab.com/runner/
- https://youtu.be/-CyVpfDQAG0?si=5E2AzxQf4dbc9bzv
- https://helm.sh/docs/intro/install/
- https://kubernetes.io/docs/tasks/administer-cluster/namespaces/
- https://man7.org/linux/man-pages/man7/dracut.cmdline.7.html
- https://bugzilla.redhat.com/show_bug.cgi?id=1205218

Q&A!

GDO Access Tracker

Creation & Deployment

HPC Cluster Engineering Academy 2024

August 7, 2024

Alicia Gullon and Kendall Parry

Lawrence Livermore National Laboratory

The Team

Alicia Gullon

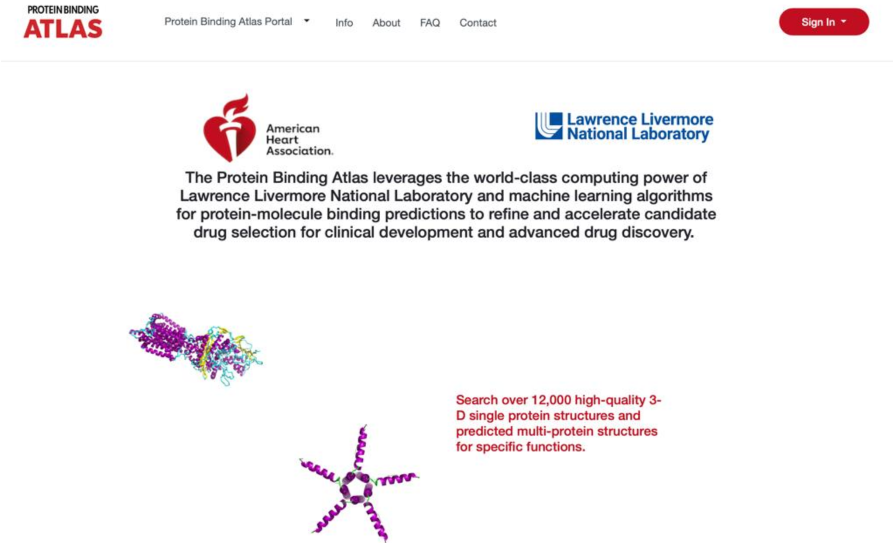
UC Berkeley

Kendall Parry

University of Puget Sound

Problem

- There is currently no convenient way for scientists to monitor the consumption of their GDO sites
  - Green Data Oasis (GDO) is used by LLNL scientists to share their data with external collaborators
  - They must reach out to Jeff Long who reviews server logs and provides a summary of the data


PROTEIN BINDING
ATLAS
Protein Binding Atlas Portal
Info
About
FAQ
Contact
Sign In
American Heart Association.
Lawrence Livermore National Laboratory

The Protein Binding Atlas leverages the world-class computing power of Lawrence Livermore National Laboratory and machine learning algorithms for protein-molecule binding predictions to refine and accelerate candidate drug selection for clinical development and advanced drug discovery.

Search over 12,000 high-quality 3-D single protein structures and predicted multi-protein structures for specific functions.

GOAL:

Develop a website to present site access statistics to end-user admins

Requirements

- For a specified range of days:
  - Number of hits per day (and total over range)
  - Number of bytes retrieved per day (and total over range)
  - Number of unique IP addresses
  - List of unique IP addresses along with hostname and geolocation
  - Top 10 files/URLs along with number of hits each

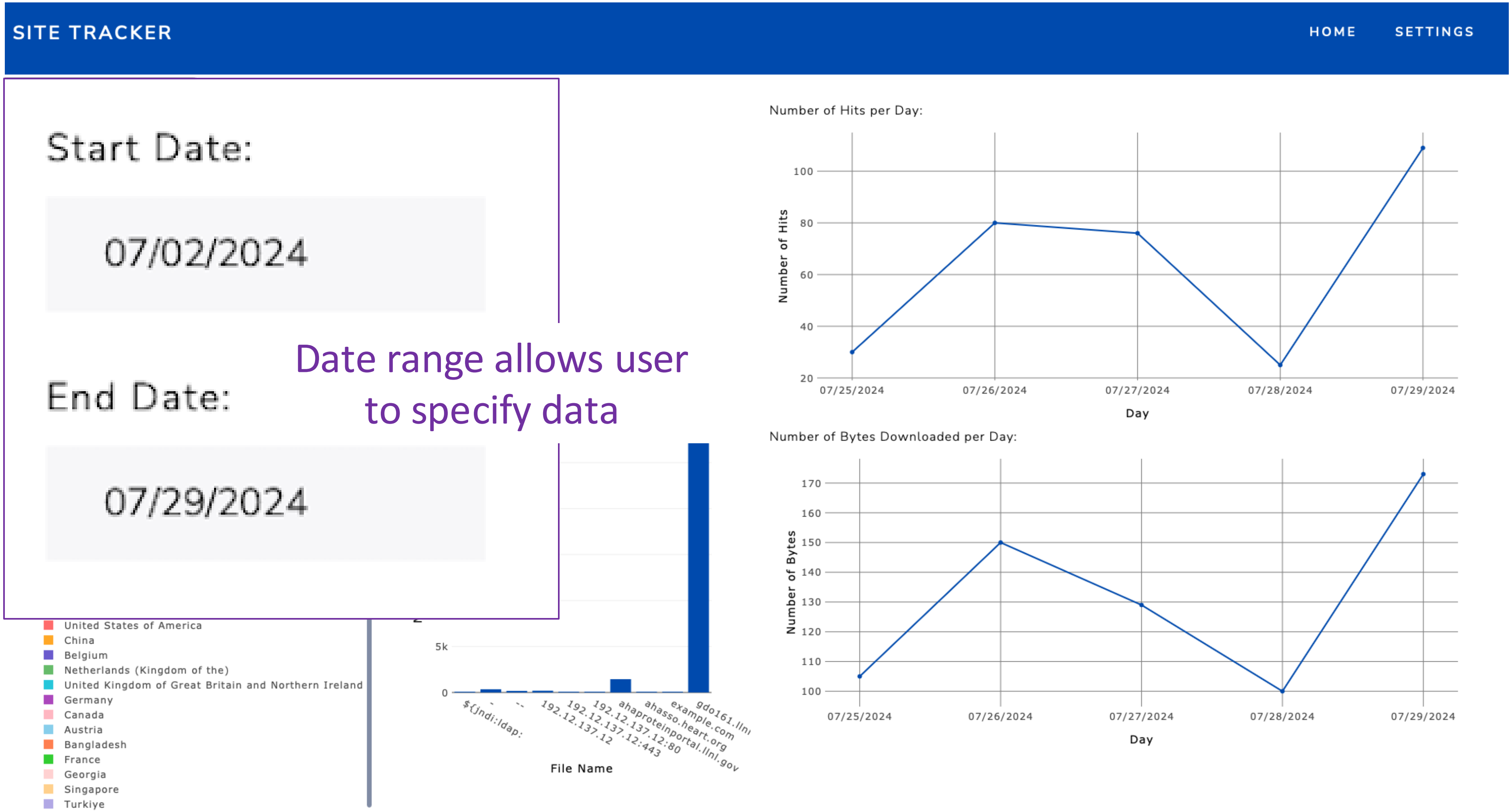
Software Stack

- Use Jinja templates and Bootstrap for design
- Plotly and JavaScript to display data
- SQLAlchemy and SQLite database to store data
- Parse data with pandas
- Configure Nginx, Gunicorn, and Flask together to deploy website

Challenges

- We do not have access to GDO clusters with the incoming logs
- Running a server through Flask limited our ability to use many common web development tools (i.e. Plotly and Bootstrap)
- Different log formats from Nginx and Apache servers
- Large amount of data to query by day

Front-End



SITE TRACKER

HOME SETTINGS

Start Date:

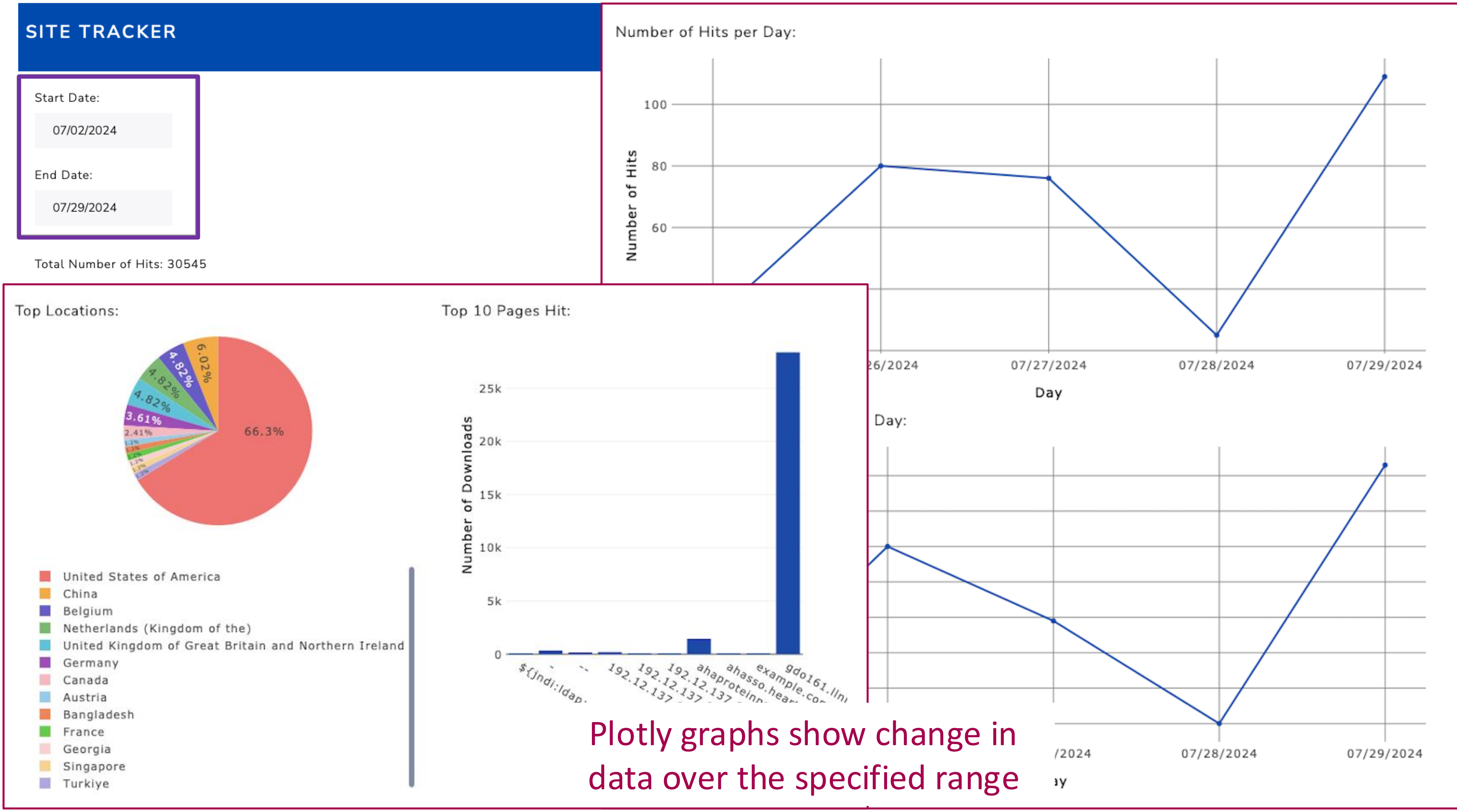
07/02/2024

End Date:

07/29/2024

Date range allows user to specify data

Front-End



SITE TRACKER

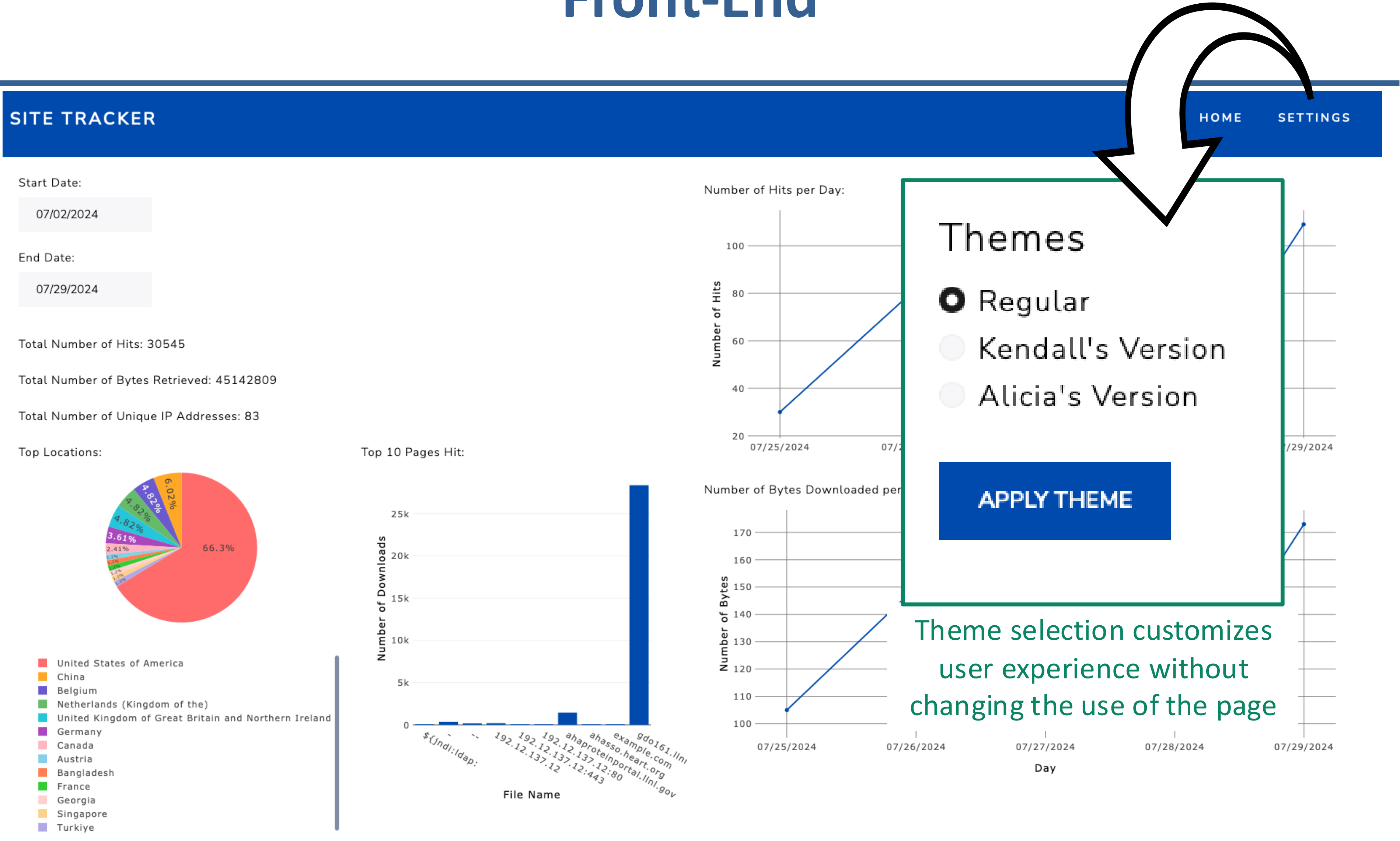
Start Date: 07/02/2024

End Date: 07/29/2024

Total Number of Hits: 30545

Plotly graphs show change in data over the specified range

Front-End



Start Date: 07/02/2024

End Date: 07/29/2024

Total Number of Hits: 30545

Total Number of Bytes Retrieved: 45142809

Total Number of Unique IP Addresses: 83

Top Locations:

Top 10 Pages Hit:

Number of Hits per Day:

Number of Bytes Downloaded per

Themes

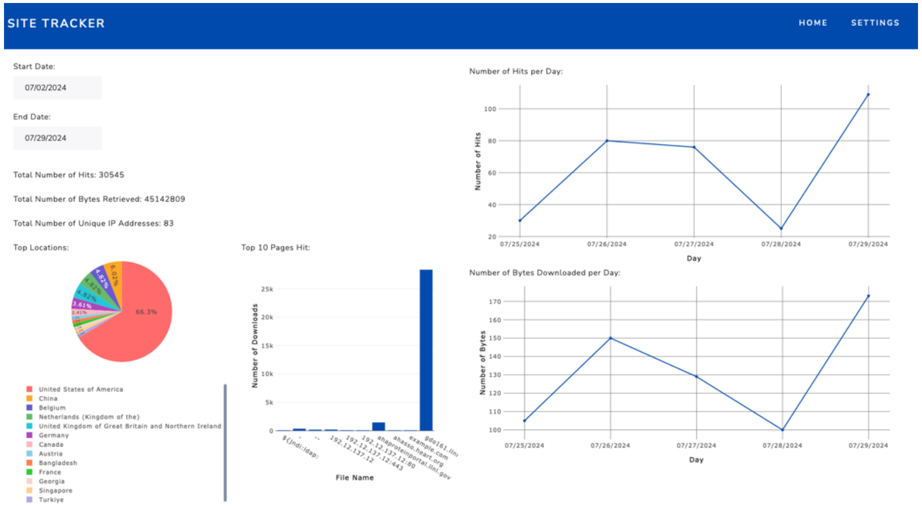
● Regular

○ Kendall's Version

○ Alicia's Version

Theme selection customizes user experience without changing the use of the page

Front-End



SITE TRACKER HOME SETTINGS

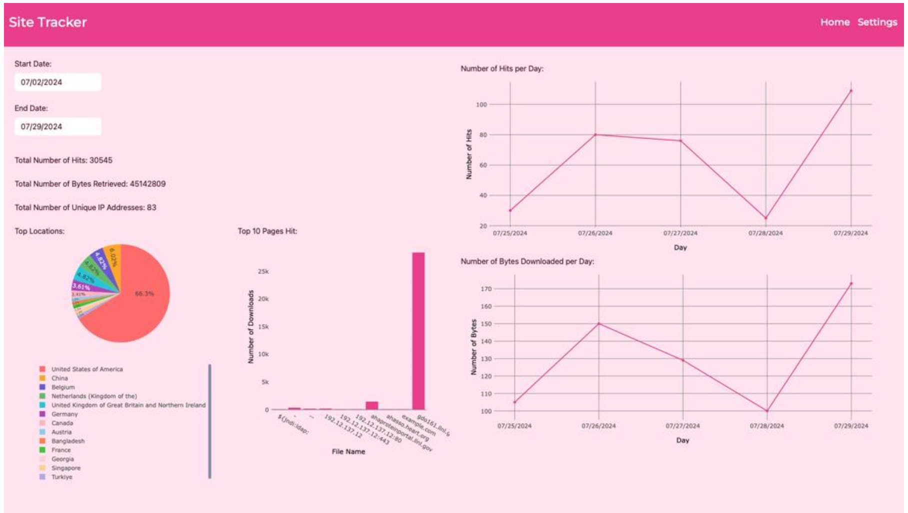
Start Date: 07/02/2024

End Date: 07/29/2024

Total Number of Hits: 30545

Total Number of Bytes Retrieved: 45142809

Total Number of Unique IP Addresses: 83



Site Tracker Home Settings

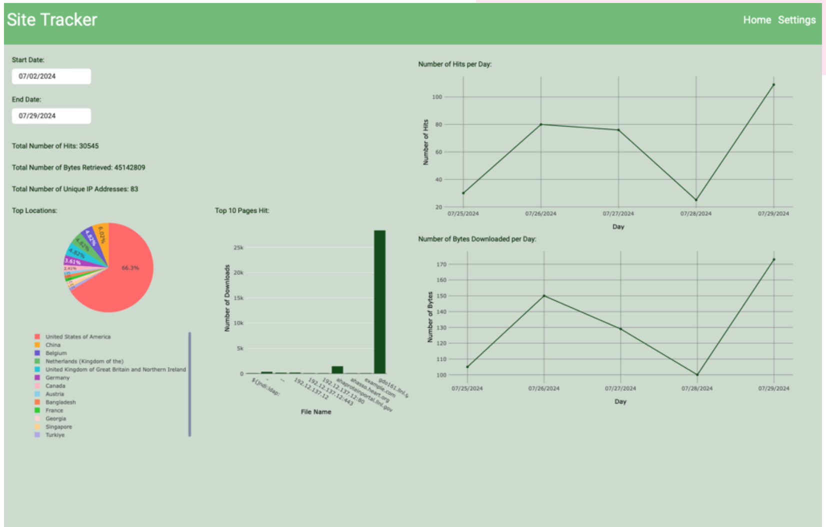
Start Date: 07/02/2024

End Date: 07/29/2024

Total Number of Hits: 30545

Total Number of Bytes Retrieved: 45142809

Total Number of Unique IP Addresses: 83



Site Tracker Home Settings

Start Date: 07/02/2024

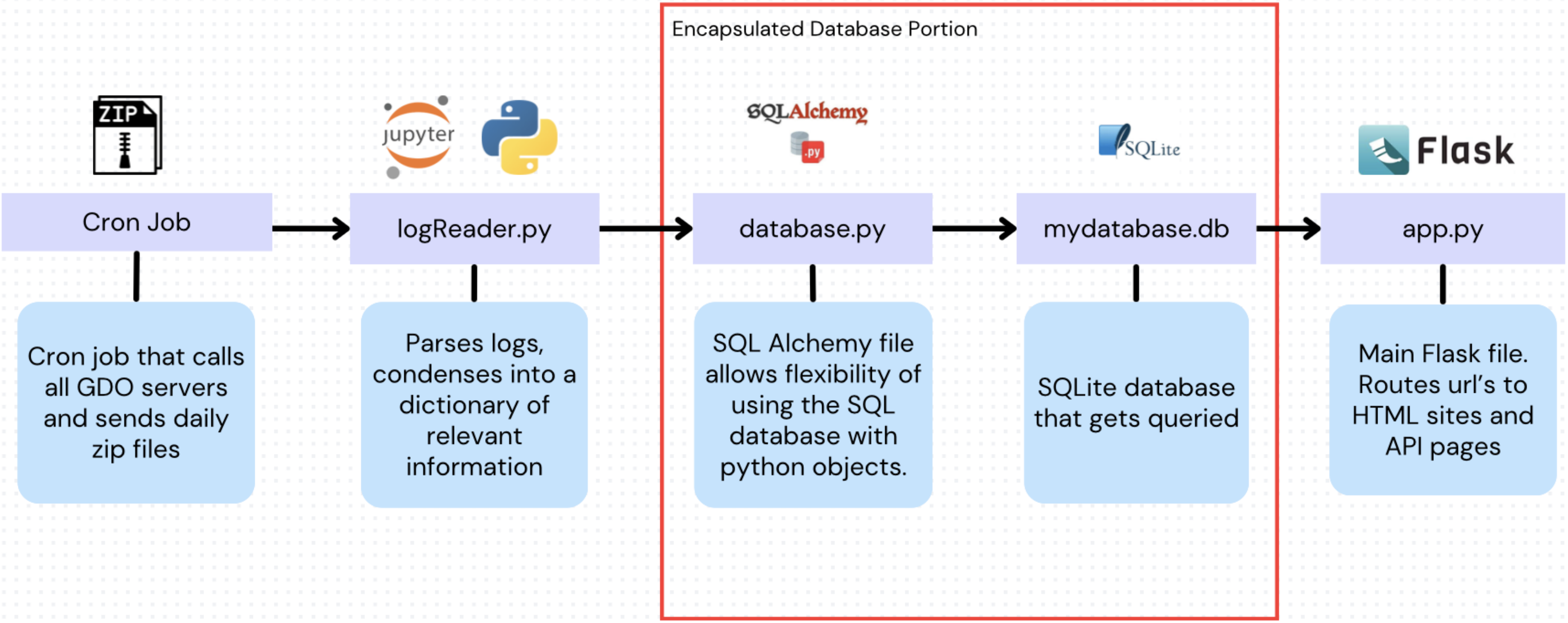
End Date: 07/29/2024

Total Number of Hits: 30545

Total Number of Bytes Retrieved: 45142809

Total Number of Unique IP Addresses: 83

Back-End Architecture



Encapsulated Database Portion

Cron Job

Cron job that calls all GDO servers and sends daily zip files

logReader.py

Parses logs, condenses into a dictionary of relevant information

database.py

SQL Alchemy file allows flexibility of using the SQL database with python objects.

mydatabase.db

SQLite database that gets queried

app.py

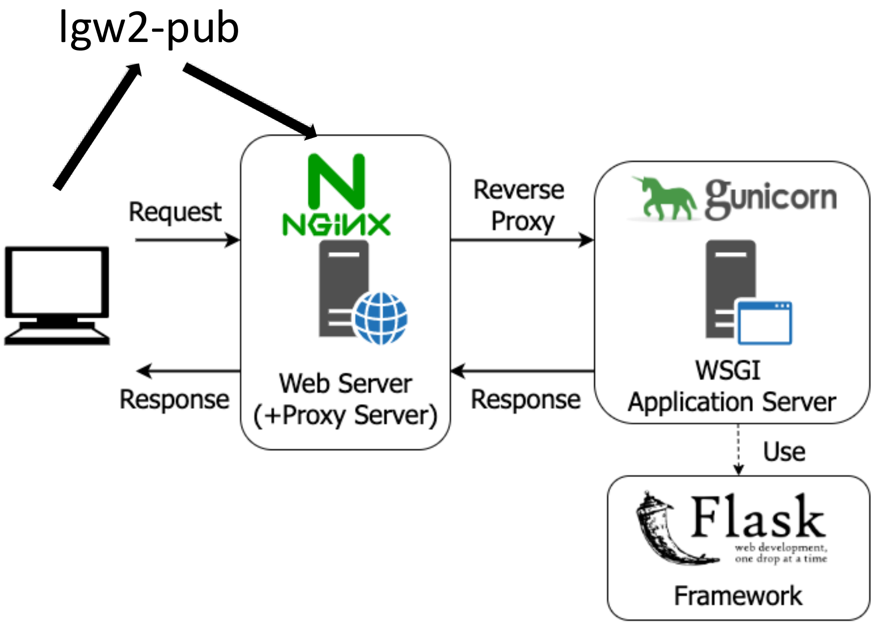
Main Flask file. Routes url's to HTML sites and API pages

Integration

- Two data presentation methods
  - Textual Display
    - **Process:** Pull data from the REST API and insert it into the HTML document
  - Graphical Display
    - **Process:** Pull data from the REST API, extract and separate keys and values, insert data into Plotly graphs
- Accurately update data on initial load and when the date range is modified

Deployment

- Localhost computer sends request to access website
- SSH tunneling allows request to go through lgw2 to Nginx server on root node of cluster
- Nginx forwards request to Gunicorn through proxy_pass
- Gunicorn handles requests and creates worker processes to run flask app
- Reverse Proxy: regulates traffic coming IN to a network, there to protect servers and for load balancing



Future

- Create additional pages with further information
  - It may be helpful to provide the users with the option of more detailed information, such as a list of IP addresses that have visited the site
- Differentiate between malicious requests/web scanning bots and real users
- Deploy on a container
  - Scalability and rollouts of changes and updates

Thank You!

Special Thanks:

Project Mentors:

Gabe Maxfield

Naomi Cheeves

Other Mentors: David Fox, Jason Kim, and Martin Baltezore

Emotional Support:

Hemingway

Sources

- Flask Documentation - https://flask.palletsprojects.com/en/3.0.x/
- Bootstrap - https://getbootstrap.com
- Plotly JavaScript - https://plotly.com/javascript/
- mdn web docs - https://developer.mozilla.org/en-US/docs/Web/HTTP/Cookies
- LivChat - https://livchat.llnl.gov/chat
- SQLAlchemy - https://www.geeksforgeeks.org/sqlalchemy-introduction/
- Deploying Gunicorn - https://docs.gunicorn.org/en/latest/deploy.html#nginx-configuration
- NGINX/Flask/Gunicorn deployment - https://dev.to/brandonwallace/deploy-flask-the-easy-way-with-gunicorn-and-nginx-jgc

Lawrence Livermore National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Trino Deployment on Clusters

HPC Cluster Engineer Academy

Lucian Chauvin and Jeremi Nuer

Mentors: Dave Fox and Jason Kim

August 7, 2024

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

Lawrence Livermore National Laboratory

About Us

Lucian Chauvin
Texas A&M University

Jeremi Nuer
UC Santa Barbara

Your Data is Everywhere

Your Data is Everywhere

- Object Storage (S3)

Your Data is Everywhere

- Object Storage (S3)
- SQL Databases

Your Data is Everywhere

- Object Storage (S3)
- SQL Databases
- Even Google Sheets!

Your Data is Everywhere

- Object Storage (S3)
- SQL Databases
- Even Google Sheets!

However, you want all your data to be easily accessible in one centralized location!

Trino is the Solution!

Trino is an open-source distributed SQL query engine.

Trino is the Solution!

Trino is an open-source distributed SQL query engine.

- Query multiple sources (databases, data lakes, object storage)

Trino is the Solution!

Trino is an open-source distributed SQL query engine.

- Query multiple sources (databases, data lakes, object storage)
- Fast queries with an arbitrary amount of workers

Trino is the Solution!

Trino is an open-source distributed SQL query engine.

- Query multiple sources (databases, data lakes, object storage)
- Fast queries with an arbitrary amount of workers
- Highly scalable for big data environments with an unknown goal or end state

Architecture

→ Data Flow

🟥 Object Storage

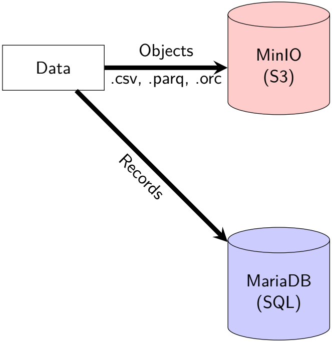
🟦 MariaDB

🟩 Trino

🟨 Exports

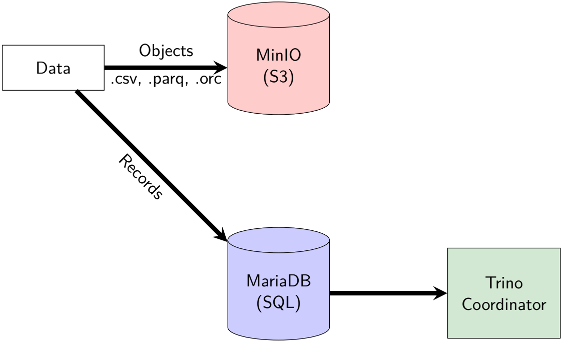
Architecture

- → Data Flow
- 🟥 Object Storage
- 🟪 MariaDB
- 🟩 Trino
- 🟨 Exports

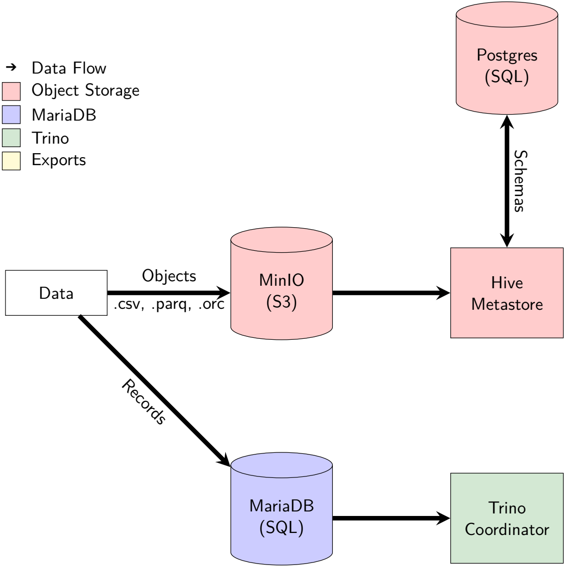


Architecture

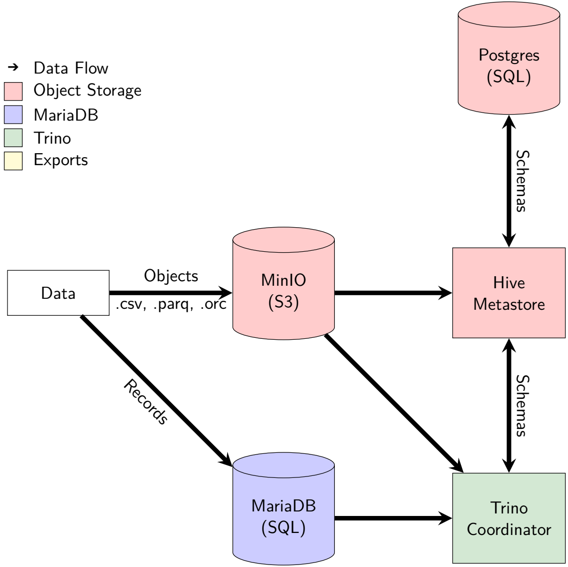
- ➔ Data Flow
- 🟥 Object Storage
- 🟦 MariaDB
- 🟩 Trino
- 🟨 Exports



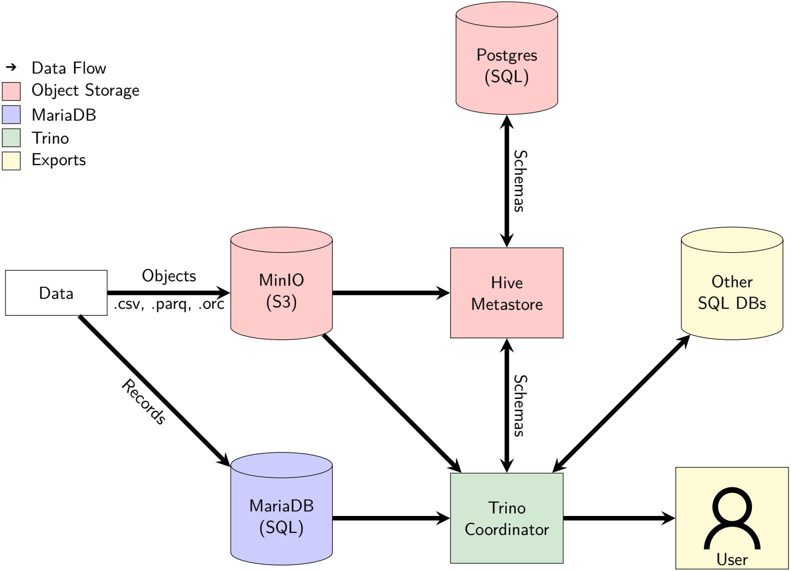
Architecture



Architecture



Architecture



➔ Data Flow

🟥 Object Storage

🟪 MariaDB

🟩 Trino

🟨 Exports

Postgres (SQL)

Schemas

Data

Objects

.csv, .parq, .orc

MinIO (S3)

Hive Metastore

Other SQL DBs

Records

Schemas

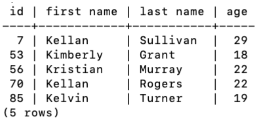
MariaDB (SQL)

Trino Coordinator

User

Use Case

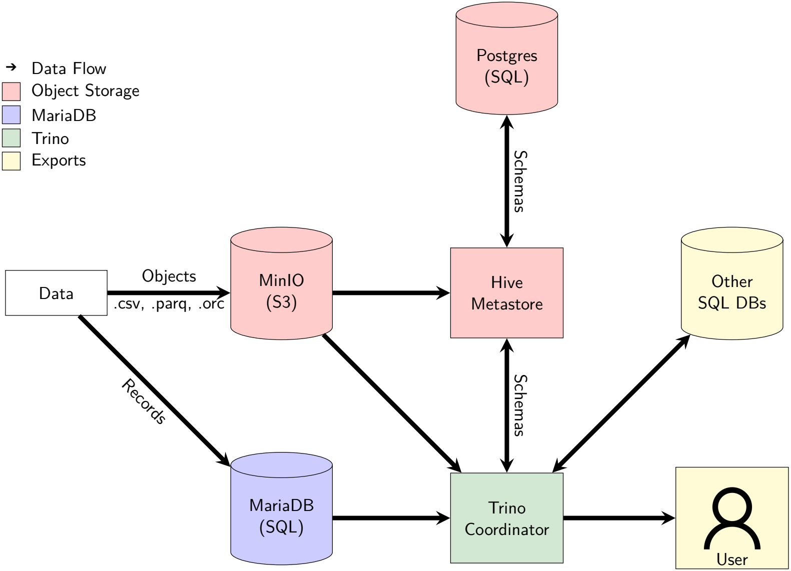
One use case that Trino enables is the ability to programmatically access, join, and write across datasources. Here is an example of this in practice:



```
 id | first name | last name | age
----+------------+-----------+-----
  7 | Kellan     | Sullivan  |  29
 53 | Kimberly   | Grant     |  18
 56 | Kristian   | Murray    |  22
 70 | Kellan     | Rogers    |  22
 85 | Kelvin     | Turner    |  19
(5 rows)
```

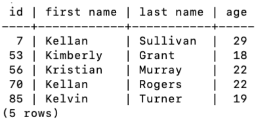
MinIO

Architecture



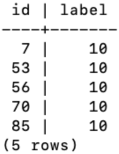
Use Case

One use case that Trino enables is the ability to programmatically access, join, and write across datasources. Here is an example of this in practice:



```
 id | first name | last name | age
----+------------+-----------+-----
  7 | Kellan     | Sullivan  |  29
 53 | Kimberly   | Grant     |  18
 56 | Kristian   | Murray    |  22
 70 | Kellan     | Rogers    |  22
 85 | Kelvin     | Turner    |  19
(5 rows)
```

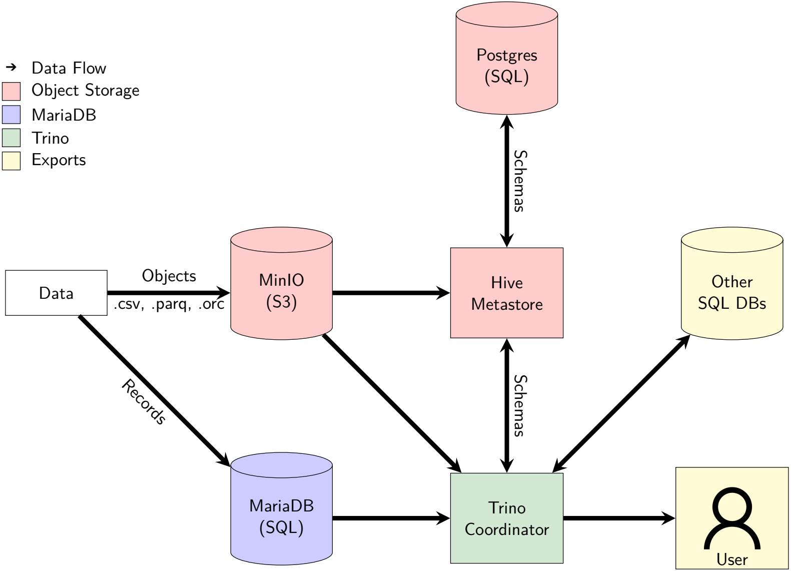
MinIO



```
 id | label
----+-------
  7 |    10
 53 |    10
 56 |    10
 70 |    10
 85 |    10
(5 rows)
```

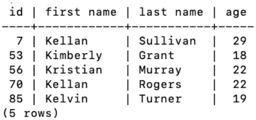
Other SQL DB

Architecture



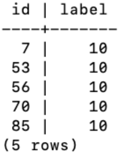
Use Case

One use case that Trino enables is the ability to programmatically access, join, and write across datasources. Here is an example of this in practice:



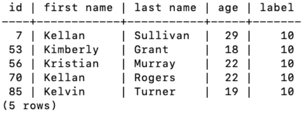
```
 id | first name | last name | age
----+------------+-----------+-----
  7 | Kellan     | Sullivan  |  29
 53 | Kimberly   | Grant     |  18
 56 | Kristian   | Murray    |  22
 70 | Kellan     | Rogers    |  22
 85 | Kelvin     | Turner    |  19
(5 rows)
```

MinIO



```
 id | label
----+-------
  7 |    10
 53 |    10
 56 |    10
 70 |    10
 85 |    10
(5 rows)
```

Other SQL DB



```
 id | first name | last name | age | label
----+------------+-----------+-----+-------
  7 | Kellan     | Sullivan  |  29 |    10
 53 | Kimberly   | Grant     |  18 |    10
 56 | Kristian   | Murray    |  22 |    10
 70 | Kellan     | Rogers    |  22 |    10
 85 | Kelvin     | Turner    |  19 |    10
(5 rows)
```

Joined

Products

- Produced a fully automated Ansible script to install and connect Trino on a cluster
- Created a python script to programmatically interact with Trino

Both of these can be found in our CZ Gitlab remote here: https://lc.llnl.gov/gitlab/chauvin3/trino-on-cluster

Challenges

- Hive Metastore installation
- Python bindings
- Database authorization across nodes using Ansible

Next Steps

- Deploy, diagnose, and compare Trino to its closed source competitors like Starburst
- Look into utilizing Trino in a ML workflow
- Test other Trino connectors

Sources

- MariaDB - https://mariadb.com/kb/en/getting-installing-and-upgrading-mariadb/
- MinIO - https://min.io/docs/minio/linux/index.html
- MinIO CLI (MC) - https://min.io/docs/minio/linux/reference/minio-mc.html
- Trino - https://trino.io/docs/current/installation/deployment.html
- trino-python-client - https://github.com/trinodb/trino-python-client

Made with LaTeX

Thank you. Any questions?

Lawrence Livermore
National Laboratory

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC