Interactive HPC using Containers on Nautilus

Introduction to PRP Nautilus, Kubernetes, and Containerized Software

Jeffrey Weekley  
24 FEB 2022
Agenda

1. Introduction
2. Background
   2.1. Pacific Research Platform
   2.2. Nautilus Cluster
3. About Containers and Kubernetes
4. Accessing Nautilus
5. Software Tools
6. Cluster Control
7. Data
8. Other considerations
9. Jupyter
10. Concluding Q&A
What is the Pacific Research Platform?

• The PRP is a partnership of more than 50 institutions, led by researchers at UC San Diego and UC Berkeley.

• PRP is an end-to-end high-speed data freeway built on CENIC and Pacific NW GigaPOP fiber optic networks

• Built for data intensive science collaboration

• Led to the rise of data-intensive science collaborations

The PRP 2015-2020
What is Nautilus?

- Nautilus1 is a highly distributed, but centrally managed community cyberinfrastructure. It is an on-demand, real-time accessible cluster with >500 GPUs and >7000 CPU cores.

- Approximately 187 nodes provide service from all 10 UC campuses and many partner institutions

- Researchers on more than 30 campuses use Nautilus

- Nautilus advantageously exploits the networking, space, and features of many of the Science DMZs previously built with NSF funding on over 100 US campuses.

- Unlike traditional HPC, Nautilus is interactive
About Containers and Orchestration Schemes...
First...what is a container?

Containerization is the packaging of software code with just the operating system (OS) libraries and dependencies required to run the code to create a single lightweight executable.

From https://www.scalyr.com/blog/containers-benefits-and-making-a-business-case
Software Containers versus Virtual Machines

Containers are an abstraction at the app layer that packages code and dependencies together. Multiple containers can run on the same machine and share the OS kernel with other containers, each running as isolated processes in user space.

Virtual machines (VMs) are an abstraction of physical hardware turning one server into many servers. The hypervisor allows multiple VMs to run on a single machine. Each VM includes a full copy of an operating system, the application, necessary binaries and libraries - taking up tens of GBs.

From https://www.docker.com/resources/what-container
Kubernetes Cluster vs. Traditional HPC

• Both K8s and HPC schedulers are workload managers

• HPC focuses on high throughput jobs with distributed memory

• Jobs are scheduled and resources can be allocated based on sophisticated accounting schemes

• K8s allows for interaction with running jobs (pods)

• K8s scheduler matches pods to appropriate nodes, more sophisticated scheduling is possible, but atypical
What is Kubernetes (K8s)

K8s is an open-source container orchestration system for automating software deployment, scaling, and management.

Key K8s Concepts

- **Control Plane** manages the workload, handles internode communications, keeps knowledge of the cluster state, provides scheduling services, manages resources, and exposes the K8s API so that both internal and external interfaces can be used.

- **Nodes** - a node is a physical machine were containers are deployed. Each node must run a container runtime (in Nautilus, it is ‘Containerd’).

- **Namespaces** provides a way for K8s to partition cluster resources across multiple or many users in an exclusive way.

- **Pods** - pods are the basic scheduling unit of K8s. Pods consist of one or more containers running inside. Each pod gets a unique IP address (important!) so that micro services or applications can access the pod without contention. Pod IP addresses are ephemeral.

- **Services** are a set of pods working together.
Pods in Nautilus

**Pods** are the smallest, most basic deployable objects in Kubernetes. A Pod represents a single instance of a running process in your cluster. Pods contain one or more containers, such as Docker containers. Pods also contain shared networking and storage resources for their containers:

- **Network:** Pods are automatically assigned unique IP addresses. Pod containers share the same network namespace, including IP address and network ports. Containers in a Pod communicate with each other inside the Pod on localhost.
- **Storage:** Pods can specify a set of shared storage volumes that can be shared among the containers.

Pods running on a cluster are automatically able to communicate with other pods.
# Managing Users with Namespaces

Groups of users working together

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Software</th>
<th>PI</th>
<th>Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>avis-citizenscience</td>
<td>UC Santa Cruz</td>
<td>Python, Tensorflow</td>
<td>Alex Pang</td>
<td></td>
</tr>
</tbody>
</table>

**Namespace**

**Description**

Project Name: A Platform for Mobile Citizen Science Apps with Client-side Machine Learning. In this project, we are developing an open-source software platform that allows a domain researcher to easily create a citizen science mobile app with client-side integrated machine learning models for collecting data with real-time analysis. The apps created with our platform can help the participants with machine learning enhanced guidance to recognize the correct data and increase the efficiency of the data collection process.

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Software</th>
<th>PI</th>
<th>Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>avis-fire</td>
<td>UC Santa Cruz</td>
<td>Python</td>
<td>Alex Pang</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Wildfire analysis with ML approaches.

<table>
<thead>
<tr>
<th>Name</th>
<th>Institution</th>
<th>Software</th>
<th>PI</th>
<th>Publications</th>
</tr>
</thead>
<tbody>
<tr>
<td>avis-rip</td>
<td>UC Santa Cruz</td>
<td>Tensorflow, Python</td>
<td>alex pang</td>
<td></td>
</tr>
</tbody>
</table>

**Description**

Deep Learning-Based Rip Current Detection: In this project, we train fully supervised deep learning object detectors to detect rip currents.
Two kinds of Nautilus Nodes

- Built on top of PRP 10-100Gbps networks
- Standardized hardware profiles
  - 8-GPU Servers (FIONA8s)
  - High-density Storage Nodes
  - NVMe and SSDs
  - 10Gbps NIC for GPUs
  - Dual 100 Gbps NIC for storage
Software Tools
Cookies
An Analogy for Nautilus

- The ingredients you choose are you code. You can adjust them to suit your needs or taste
- The mixer (or spoon and bowl) are your development tools (e.g. Docker or VIM)
- The YAML file is the recipe. How you will mix your ingredients together
- kubectl controls the oven
- Nautilus is your oven. It has many features to help you make your cookies delicious.
Using, Creating and Deploying Containerized Software
Docker, Docker Hub, YAML and Kubectl

• Docker (or other containerization software) is used to create images. Common practice is to develop inside Docker, with the container running while you make changes to your code (aka DevOps)

• Docker Hub contains thousands of pre-built images (what you need may already be available)

• Once you know what software you want to use, YAML provides the “recipe” for your deployment

• Kubectl is the tool to launch your deployments
Let’s examine Docker...
We know that…

- Kubernetes is an orchestration framework for deploying containerized software. It is naive about the actual software.
- In order for you to develop Services, you will need to use or create software to deploy
- Most people can simply re-use software by referencing a pre-built Docker image
- Users customize existing Docker images with additional software for their purpose
- There are many thousands of these images available on Docker Hub
- Users can create their own software containers using Docker on their local computer
Container Supply Chain
aka DevOps

Docker Hub

Public Image Repository

Fetch Base Image

Version Control

Build & Test Automation

Build Final Image

Push Final Image

Private Image Repository

Fetch Image

Orchestrator

Deploy

Operational Environment

You

Software Engineer

Operations Engineer

kubectl + YAML

Your GitLab

Kubernetes

Nautilus

Figure 4: Model Container Supply Chain

From: CMU's Software Engineering Institute "7 Quick Steps to Using Containers Securely"
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: mypod
    image: centos:centos7
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: 
      - sh
      - -c
      - echo 'I am a new pod' && sleep infinity

YAML: Yet Another Markup Language
Code is what you will run...

Using YAML and kubectl is how you will run it
What is YAML?

- A human-readable data-serialization language
- Commonly used for configurations
- Similar to eXtensible Markup Language, but with fewer syntax conventions
- Python-style indentation to reflect hierarchy and order-of-operations
- Encodes strings, integers, floats and arrays
- Most Nautilus users modify example YAML files
- Used to request resources, deploy services and code (in containers)
Sample YAML File

for a generic pod

apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: mypod
    image: centos:centos7
    resources:
      limits:
        memory: 100Mi
        cpu: 100m
      requests:
        memory: 100Mi
        cpu: 100m
    command: ["sh", "-c", "echo 'I am a new pod' && sleep infinity"]
Using Nautilus

Account setup
Namespace
Software
Deployment
Getting Help
Nautilus Access Levels

There are three levels of access to Nautilus:

- Guest
- User
- Admin
Logging in for the First Time
Using CI Login and Your Institutional Credentials

PRP Kubernetes portal
Here you can get an account in Pacific Research Platform kubernetes portal by logging in with your university’s credentials and requesting access in [matrix]

Documentation: http://ucsd-prp.gitlab.io/userdocs/

You can easily join your node in our cluster - request instructions in [matrix] #general channel.
Guest Level

Once you’ve logged in for the first time, you are a “Guest”

Guests have credentials but no privileges. Before you can access resources, you must be validated.

At this stage, you merely get your system configured to use Nautilus
Required Software and Configuration

Kubectl and Nautilus Configuration File

Support channel
We provide support in [matrix]: https://ucsd-prp.gitlab.io/userdocs/start/contact/

Quick Start guide
http://ucsd-prp.gitlab.io/userdocs/start/quickstart/

YOU'RE REQUIRED TO READ THE
>>>>>>>>>> DOCUMENTATION <<<<<<<<<<<
BEFORE RUNNING ANYTHING IN THE CLUSTER
In your home directory, create a hidden folder, where you will place the downloaded configuration file.
The Configuration File

Contains credentials and configurations that link your local computer to the cluster
Access Levels

User

Admin

Guest
User Level

Guests navigate to Nautilus’ Matrix Chat #Nautilus Support channel to be promoted from Guest to User.

Users may then join an existing namespace by submitting a request to the namespace admin.
Admin Level

• Users may request to be promoted to Admin in the chat support channel

• Admins may create namespaces, join users to them and are responsible for maintaining community standards for the users in their namespaces
Requesting Privileges (and Getting Help)

Via the Matrix chat at https://element.nrp-nautilus.io

All requests for assistance are done in Matrix
Controlling the Cluster from your Desktop
Review of what we’ve done so far…

• We have access to Nautilus and we’ve been promoted to User (to join someone else’s namespace) or we’ve created our own namespace as Admin
• We have installed Docker so we can build or modify images (i.e. containers)
• We have access to a private repo
• We have installed our config file so the cluster knows who we are

How do we actually control the cluster?
Installing Kubectl

In order to use your config file to control the cluster, you will need to install this tool.

The Kubernetes command-line tool, `kubectl`, allows you to run commands against Kubernetes clusters. You can use `kubectl` to deploy applications, inspect and manage cluster resources, and view logs. For more information including a complete list of `kubectl` operations, see the `kubectl` reference documentation.

Installation instructions can be found here:
https://kubernetes.io/docs/tasks/tools/
Structure of kubectl

Users interface with a Kubernetes-controlled system using The Kubernetes command-line tool, “kubectl”

kubectl [command] [TYPE] [NAME] [flags]

“command” describes the operation to perform. They are limited to these operations:

- create generates new resources from files or standard input devices
- describe retrieves details about a resource or resource group
- get fetches cluster data from various sources
- delete removes resources as directed
- apply pushes changes based on configuration files
Structure of kubectl (cont)

Users interface with a Kubernetes-controlled system using The Kubernetes command-line tool, “kubectl”

```
kubectl [command] [TYPE] [NAME] [flags]
```

[TYPE] specifies the category of resource you are targeting

[NAME] this case-sensitive field specifies the name of the resource (you will name your pods, deployments, etc.) Using the NAME field restricts to operation to that named resource. Leaving it blank will apply it universally.

[Flags] the modifier [Flags] denotes any special options or requests made of a resource. This modifier can be used to over-ride defaults or environment variables.
What about my data?
Persistent Data in K8s

- Managing storage is a distinct problem from managing compute instances
- Storage is provisioned through the PersistentVolumeClaim, on Nautilus comes in distinct classes and regions
- Ceph shared filesystem (CephFS) is the primary way of storing data in Nautilus which allows mounting the same volume(s) from multiple PODs in parallel

<table>
<thead>
<tr>
<th>StorageClass</th>
<th>Filesystem Type</th>
<th>Region</th>
<th>AccessModes</th>
<th>Storage Type</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>rook-cephfs</td>
<td>CephFS</td>
<td>US West</td>
<td>ReadWriteMany</td>
<td>Spinning drives with NVME meta</td>
<td>2.5 PB</td>
</tr>
<tr>
<td>rook-cephfs-east</td>
<td>CephFS</td>
<td>US East</td>
<td>ReadWriteMany</td>
<td>Mixed</td>
<td>1 PB</td>
</tr>
<tr>
<td>rook-cephfs-pacific</td>
<td>CephFS</td>
<td>Hawaii+Guam</td>
<td>ReadWriteMany</td>
<td>Spinning drives with NVME meta</td>
<td>384TB</td>
</tr>
<tr>
<td>beegfs</td>
<td>BeeGFS</td>
<td>US West</td>
<td>ReadWriteMany</td>
<td></td>
<td>2PB</td>
</tr>
<tr>
<td>rook-ceph-block</td>
<td>RBD</td>
<td>US West</td>
<td>ReadWriteOnce</td>
<td>Spinning drives with NVME meta</td>
<td>2.5 PB</td>
</tr>
<tr>
<td>rook-ceph-block-east</td>
<td>RBD</td>
<td>US East</td>
<td>ReadWriteOnce</td>
<td>Mixed</td>
<td>1 PB</td>
</tr>
<tr>
<td>rook-ceph-block-pacific</td>
<td>RBD</td>
<td>Hawaii+Guam</td>
<td>ReadWriteOnce</td>
<td>Spinning drives with NVME meta</td>
<td>384TB</td>
</tr>
<tr>
<td>seaweedfs-storage</td>
<td>SeaweedFS</td>
<td>US West</td>
<td>ReadWriteMany</td>
<td>NVME</td>
<td>300 TB</td>
</tr>
</tbody>
</table>
Other Types of Storage in Nautilus

• Local: Most nodes in the cluster have local NVME drives, which provide faster I/O than shared filesystems. These can be used for workloads that require very intensive I/O operations

• Nextcloud: access to the Nextcloud instance running on Nautilus. It's similar to other file sharing systems (Dropbox, Google Drive etc) and can be used to get data in the cluster, temporary stage the results, share data and so on

• SeaweedFS is a high-performance distributed filesystem, optimized for working with huge number of files and also huge files

• SyncThing is a tool to synchronize files collections between several devices with no single server, which creates a mesh between all devices and works well for large files collections
Moving Data

Remember: K8s pods have local addresses and are not accessible from outside

- Kubectl has a copy function:

  kubectl -n my_namespace cp ~/tmp/file.dat my_super_pod:/tmp/file.dat

  This method is not suitable for large data transfers!

- Using S3 is the most scaleable way to move large data sets. Refer to the [S3 Documentation](#) for more information

- Directly manipulating from inside your pod using standard Linux tools. This allows you to move the data in as if it were another physical host. All the same rules apply. CAVEAT: once the pod goes away, so does your data.
Other considerations...
Microservices

Breaking your workflow into services, and those services into microservices may help you debug and scale.

What is a Microservice?

- A microservice is an architectural design for building a distributed application.

- Microservices break an application into independent, loosely-coupled, individually deployable services.

- This architecture allows for each service to scale or update using the deployment of service proxies without disrupting other services in the application and enables the rapid, frequent and reliable delivery of large, complex applications.
## Additional Applications

Some helpful applications run natively in Nautilus

<table>
<thead>
<tr>
<th>Computation</th>
<th>Collaboration</th>
<th>Monitoring</th>
</tr>
</thead>
<tbody>
<tr>
<td>• JupyterHub (West Coast)</td>
<td>• EtherPad: notebooks</td>
<td>• Traceroute tool</td>
</tr>
<tr>
<td>• JupyterHub (East Coast)</td>
<td>• GitLab</td>
<td>• PerfSONAR</td>
</tr>
<tr>
<td>• WebODM (Web Open Drone Map): Drone Images stitching</td>
<td>• Jitsi: Video conferencing</td>
<td></td>
</tr>
<tr>
<td>• Appwrite: Backend Server for Web, Mobile, others</td>
<td>• Nextcloud: File sharing</td>
<td></td>
</tr>
<tr>
<td></td>
<td>• Overleaf: LaTeX collaboration</td>
<td></td>
</tr>
</tbody>
</table>
More on Jupyter Lab
Jupyter on Nautilus

A few things to know...

- New users should request access in Matrix

- Multiple GPU types available:

<table>
<thead>
<tr>
<th>GPU Type</th>
<th>GPU Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>1060</td>
<td>1080</td>
</tr>
<tr>
<td>1080Ti</td>
<td>2080Ti</td>
</tr>
<tr>
<td>TITAN XP</td>
<td>Tesla K40</td>
</tr>
<tr>
<td>TITAN RTX</td>
<td>3090</td>
</tr>
<tr>
<td>Tesla V100</td>
<td>RTX A100</td>
</tr>
<tr>
<td>RTX8000</td>
<td>RTX A40</td>
</tr>
</tbody>
</table>
Jupyter on Nautilus

A few more things to know…

- Mounting of CephFS PVCs allowed
- Preconfigured, common software stacks
- Runs common iPython Notebooks
- Notebooks persist (very handy)
**Important note:** You should always work on a duplicate of the course notebook. On the page you used to open this, tick the box next to the name of the notebook and click duplicate to easily create a new version of this notebook.

You will get errors each time you try to update your course repository if you don't do this, and your changes will end up being erased by the original course version.

```python
[11]: import matplotlib.pyplot as plt
```

**Welcome to Jupyter Notebooks!**

```python
[12]: plt.plot([1, 2, 3], [2, 4, 3])
    plt.show()
```

If you want to learn how to use this tool you've come to the right place. This article will teach you all you need to know to use Jupyter Notebooks effectively. You only need to go through Section 1 to learn the basics and you can go into Section 2 if you want to further increase your productivity.
Review

Putting it all together...

• There are software tools and configurations you’ll need: Docker, kubectl, config file in ~./kube directory
• Docker can be used to build customize containers
• Docker Hub has many pre-built software containers
• Access to Nautilus is granted at three levels: guest, user and admin
• kubectl is how you interact with the cluster
• Your YAML file tells the cluster what you want it to do
• Deployments are groups of pods and services that make up your workflow
• Storage outside of your pod is available through a PersistentVolumeClaim
• Jupyter Lab is a good place to start exploring
Final Q&A

Questions about this Nautilus: see Documentation

Questions about this presentation? jweekley@ucsc.edu

For operational support, visit Nautilus Matrix chat
The End

Introduction to PRP Nautilus, Kubernetes, and Containerized Software