



FUZZBALL

Modern Performance Computing,
Now Available On-Prem

This white paper provides a technical overview of the main concepts that comprise the Fuzzball on-premise platform as it exists in Q3 2024. The paper also aims to demonstrate how Fuzzball eschews many of the established patterns, behaviors, and expectations of traditional HPC environments. For those new to HPC, the concepts of Fuzzball may not appear to be novel, but for those with deep experience in HPC environments and workloads, Fuzzball offers a paradigm shift away from designs that are 30+ years old. For a high-level overview of Fuzzball, please see [the Fuzzball product page at CIQ.com](#).

By Jonathon Anderson, Principal HPC Engineer at CIQ

Fuzzball is a modern, container-first performance computing platform. Its central management system, Fuzzball Orchestrate, schedules and dispatches compute workflows. Fuzzball Substrate, its execution agent, performs storage management, prepares container images, and executes computational work in a custom container runtime built for maximum performance in both single- and multi-node performance computing use cases.

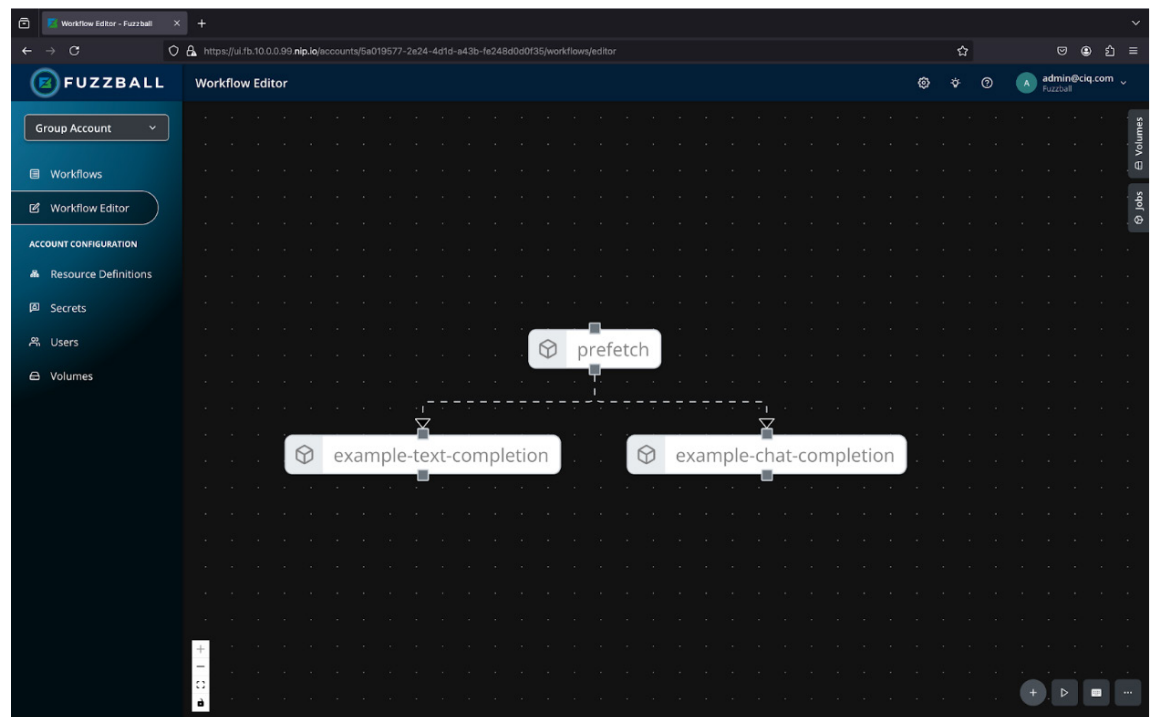
Traditional HPC systems have evolved over time in light of modern developments; but their foundation in a shell-scripted research environment continues to complicate the user experience by bringing the entire system-level Linux environment in-scope as a user-facing concern.

Fuzzball simplifies performance computing with

- native containerization support;
- transparent management of local and external storage resources;
- holistic workflows that semantically define all job environment, storage, and resource dependencies;
- and a full gamut of discoverable, accessible, and automatable user interfaces.

Feature	Slurm	Fuzzball
Web UI	partial	full
Workflows	partial	yes
Storage volume management	no	yes
Secrets management	no	yes
CLI	yes (SSH required)	yes (no SSH required)
Container management	optional	native

Workflows



Where traditional HPC systems are based primarily on shell-script-defined “batch jobs,” Fuzzball organizes work into **workflows**. A Fuzzball workflow is a structured representation of the work to be done, primarily defined by the jobs that the workflow will execute and the storage volumes that the jobs will use. This more holistic approach to computing workloads allows the system to better support multi-step workloads, dynamic runtime requirements, data access, and data movement.

Workflow jobs define

- **an execution environment:** primarily a container image for the job to execute within, but also including environment variables and other runtime parameters
- **resources required for the job:** including CPU, memory, and GPU
- **and the command(s) to execute.**

Storage volumes are identified at the workflow level and may be referenced by any job in that workflow. Any referenced workflow may, optionally, include an ingress and an egress step to copy data from or to an external HTTP(S) or S3 service (optionally with a pre-configured authentication credential). These storage preparation steps are added to the workflow automatically as additional, system-defined jobs that execute along with the rest of the workflow.

Multiple jobs in a workflow may run concurrently; or, given a defined dependency relationship between two or more jobs, may wait for a previous job to finish before starting the next job. (Other standard dependencies are configured automatically for container, ingress, and egress jobs.)

Workflow jobs can access any of the workflow's volumes by attaching them to the file system at a user-defined location. Each job configures volume attachment independently, based on the requirements and expectations of that job's process.

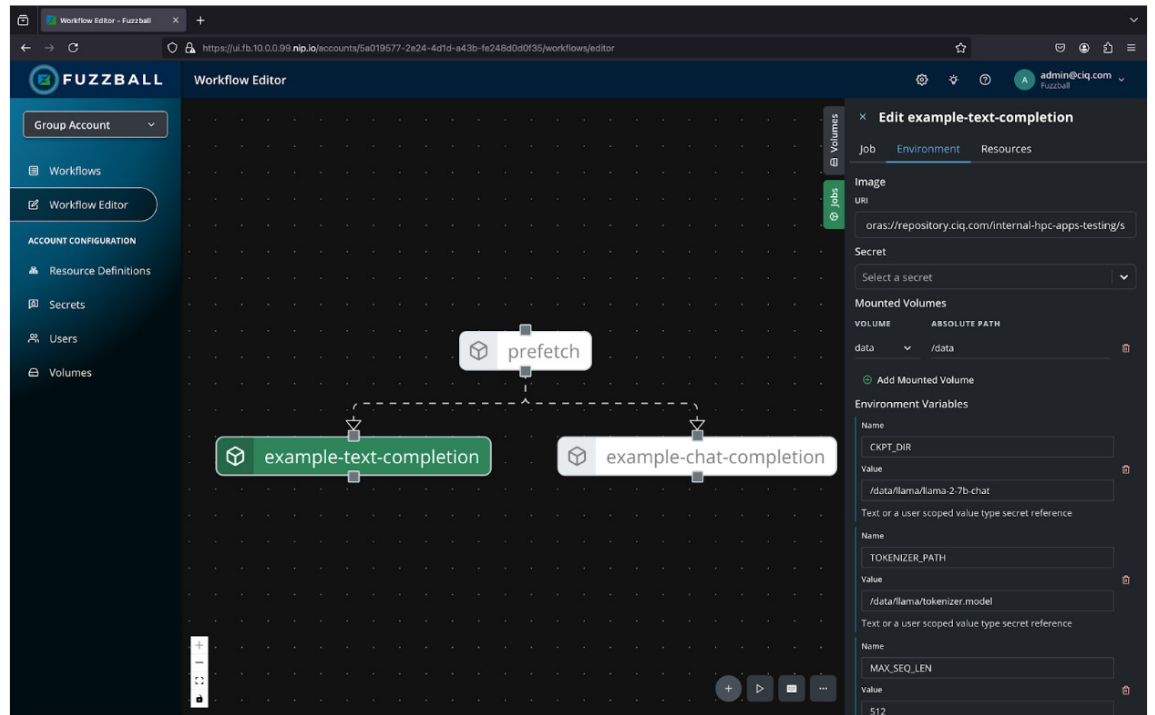
Fuzzball workflows are [commonly represented in YAML format](#), but may also be expressed in JSON format and built interactively with the Fuzzball web interface.

Example workflows can be found [in the Fuzzball documentation](#).

```
version: v1
jobs:
  mpi-hello-world:
    env:
      - PATH=/usr/lib64/mpich/bin:/usr/local/bin:/usr/bin:/bin
    image:
      uri: oras://docker.io/anderbubble/mpich-hello-world.sif
    command:
      - /bin/sh
      - '-c'
      - mpi_hello_world
    resource:
      cpu:
        cores: 4
        affinity: NUMA
      memory:
        size: 1GB
    multinode:
      nodes: 1
      implementation: mpich
```

an MPI-based "Hello, world" workflow

Job containerization and execution

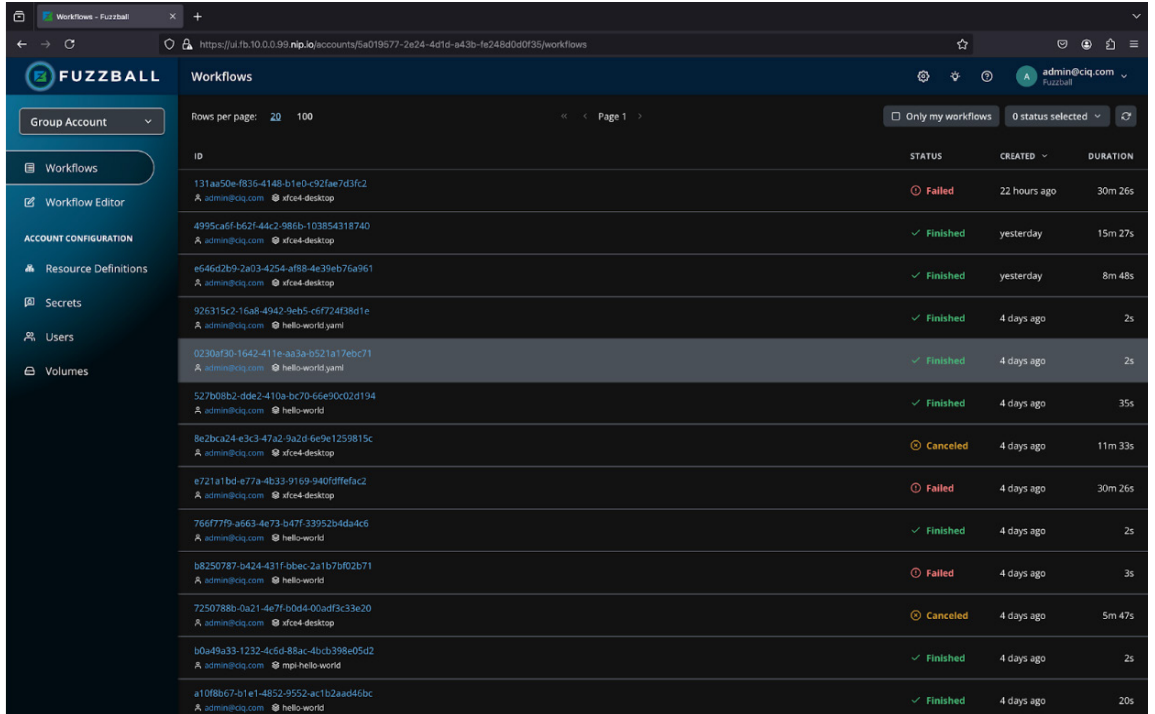


In a traditional HPC environment, user processes typically share an execution environment with the system itself, and that environment must contain all the libraries, configurations, applications and data that users of the system need. Sometimes, these requirements conflict with each other, causing increasing complexity for both end-users and system administrators.

More recently, containerization (with tools like [Apptainer](#)) have allowed for greater flexibility for performance computing by allowing an application to run in its own custom execution environment; but a lack of integration between traditional HPC tools and these container technologies continues to complicate, or even prevent, certain workload types (particularly distributed computing / MPI workloads) from being able to use containerization techniques.

Fuzzball unlocks the benefits of containerization for all workloads. Each job in a Fuzzball workflow is executed by Fuzzball Substrate, a custom container runtime built for performance computing. Fuzzball substrate supports container images in the Singularity image format (SIF) natively for best performance; but Open Container Initiative (OCI, or “Docker” style) images may be specified as well and are converted transparently to SIF format on use.

Interface



ID	STATUS	CREATED	DURATION
131aa50e-f836-4148-b1e0-c92fae7d3f2 A. admin@ciq.com @ xfce4-desktop	Failed	22 hours ago	30m 26s
4995ca6f-b621-44c2-986b-103854318740 A. admin@ciq.com @ xfce4-desktop	Finished	yesterday	15m 27s
e646d2b9-2a03-4254-a188-4e39eb76a961 A. admin@ciq.com @ xfce4-desktop	Finished	yesterday	8m 48s
926315c2-16a8-4942-9eb5-c9f724f38d1e A. admin@ciq.com @ hello-world.yaml	Finished	4 days ago	2s
0230df30-1642-411e-aa3a-b521a17ebc71 A. admin@ciq.com @ hello-world.yaml	Finished	4 days ago	2s
527b08b2-dde2-410a-bc70-f6e490c02d194 A. admin@ciq.com @ hello-world	Finished	4 days ago	35s
8e2bca24-e3c3-47a2-9a2d-6e9e1259815c A. admin@ciq.com @ xfce4-desktop	Canceled	4 days ago	11m 33s
e721a1bd-e77a-4b33-9169-940df1efac2 A. admin@ciq.com @ xfce4-desktop	Failed	4 days ago	30m 26s
766f779-a663-4e73-b47f-33952b4da4c6 A. admin@ciq.com @ hello-world	Finished	4 days ago	2s
b8250787-b424-431f-bbec-2a1b7bf02b71 A. admin@ciq.com @ hello-world	Failed	4 days ago	3s
7250788b-0a21-4e7f-b0d4-00dff3c33e20 A. admin@ciq.com @ xfce4-desktop	Canceled	4 days ago	5m 47s
b0a49a33-1232-4c6d-88ac-40cb398e05d2 A. admin@ciq.com @ mpi-hello-world	Finished	4 days ago	2s
a108b67-b1e1-4852-9552-act1b2aad46c A. admin@ciq.com @ hello-world	Finished	4 days ago	20s

Traditional HPC environments are typically based on a command-line interface deployed in a shared, user-accessible shell environment (e.g., a “login node.”) This requires that end-users not only understand the resources available, but with the complexity and seeming obscurity of a multi-user Linux environment.

Fuzzball provides three user interfaces: a discoverable web interface, a portable command-line interface, and a rich API.

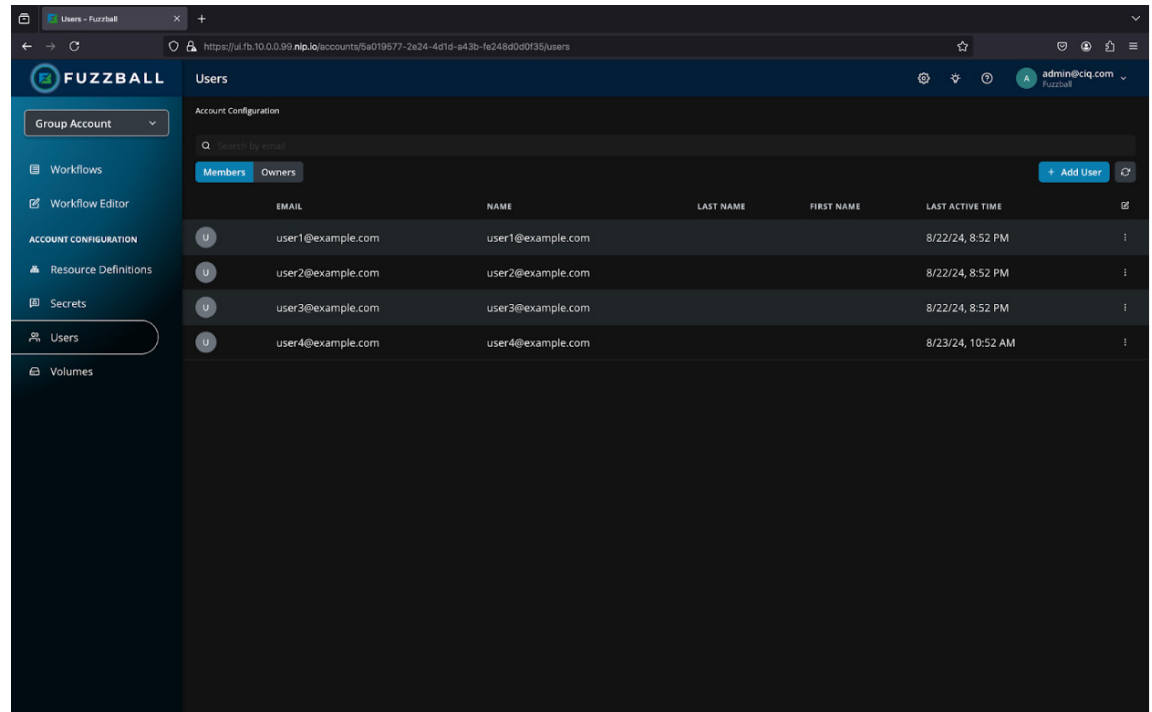
The Fuzzball web interface is accessible from any web browser and is an intuitive and discoverable way to monitor and review Fuzzball workflow status, fetch job logs, create and develop Fuzzball workflows, directly access the run-time environment of a running job, and generally monitor and manage your Fuzzball environment.

The Fuzzball command-line interface (CLI) differs from those in similar tools by being, like the web interface, backed by the Fuzzball API. The CLI can run on any system with network access to Fuzzball Orchestrate, and is typically run directly on end-user workstations without need for SSH or other shell access to an intermediate “login node.”

In addition to all the functionality of the web UI, the command-line interface also supports port-forwarding to running Fuzzball jobs across the API, allowing workflow processes (e.g., Jupyter and virtual desktops) to be accessed directly from the local workstation.

The Fuzzball API, which backs both the CLI and the Web UI, is accessible via both gRPC and OpenAPI protocols. The OpenAPI endpoint, in particular, provides a natural interface for custom site integrations. (For example, a Python SDK based on the OpenAPI endpoint is available to include Fuzzball in any automated workflow execution, CI/CD, or other process.)

Users and account management



In a traditional HPC environment, end-user accounts are system-level accounts. Users and groups must be defined for all compute nodes, login nodes, and storage nodes in the environment, and a simple misconfiguration of authorization policies could grant unintended access to back-end systems (or even just unscheduled compute resources). Fuzzball centrally manages all user accounts and implements required accounts within job containers, obsoleting system-level user accounts.

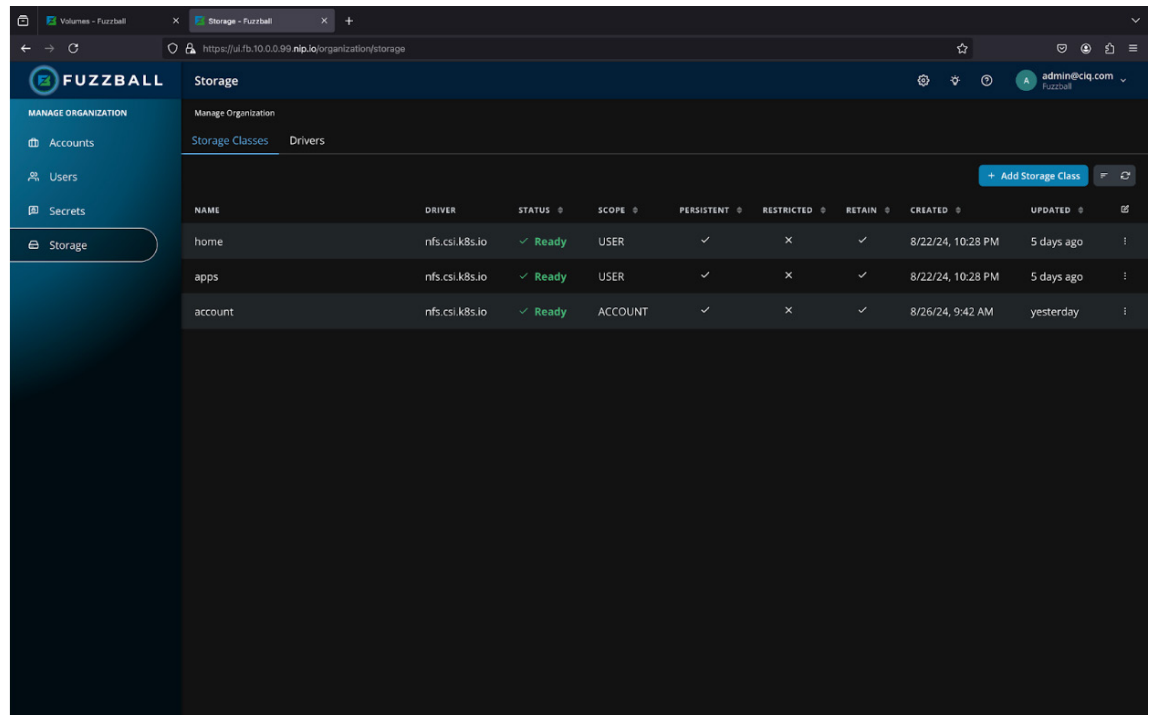
Fuzzball **users** are members of a Fuzzball **organization**. This organization is backed by a Keycloak¹ realm, which may be deployed as part of Orchestrate or accessed as an external site service.

Users are further members of one or more Fuzzball **accounts**. Fuzzball accounts allow users to collaborate on workflows, access shared storage, and securely exchange and access secrets required for proper workflow execution and resource access

¹ <https://www.keycloak.org/>

Keycloak supports integration with Active Directory and LDAP, and users defined through these integrations are accessible to Fuzzball. In addition, POSIX identities (“uid” and “gid”) can be made available to Fuzzball, and these identities are then used by Fuzzball jobs during execution. (If no POSIX identities are available from Keycloak, unique identities are created automatically for each Fuzzball user and account.)

Storage



The screenshot shows the Fuzzball web interface for managing storage. The left sidebar contains navigation options: Accounts, Users, Secrets, and Storage. The main content area displays a table of storage classes under the heading 'Storage Classes'. The table has columns for NAME, DRIVER, STATUS, SCOPE, PERSISTENT, RESTRICTED, RETAIN, CREATED, and UPDATED. There are three rows of storage classes listed.

NAME	DRIVER	STATUS	SCOPE	PERSISTENT	RESTRICTED	RETAIN	CREATED	UPDATED
home	nfs.cs.k8s.io	Ready	USER	✓	✗	✓	8/22/24, 10:28 PM	5 days ago
apps	nfs.cs.k8s.io	Ready	USER	✓	✗	✓	8/22/24, 10:28 PM	5 days ago
account	nfs.cs.k8s.io	Ready	ACCOUNT	✓	✗	✓	8/26/24, 9:42 AM	yesterday

In a traditional HPC environment, all storage is mounted persistently (or automatically) to all storage resources: all jobs have access to all storage at all times, regardless of whether they need it or not. This complicates reproducibility by obscuring dependencies between storage and application. Further, storage access is shared between user-facing application access and backend system access, increasing the risk of system-wide disruptions due to abnormal load or storage system failure.

Fuzzball accesses storage using the Container Storage Interface (CSI) originally developed for use with Kubernetes. This allows Fuzzball to use any storage that already provides a CSI driver.

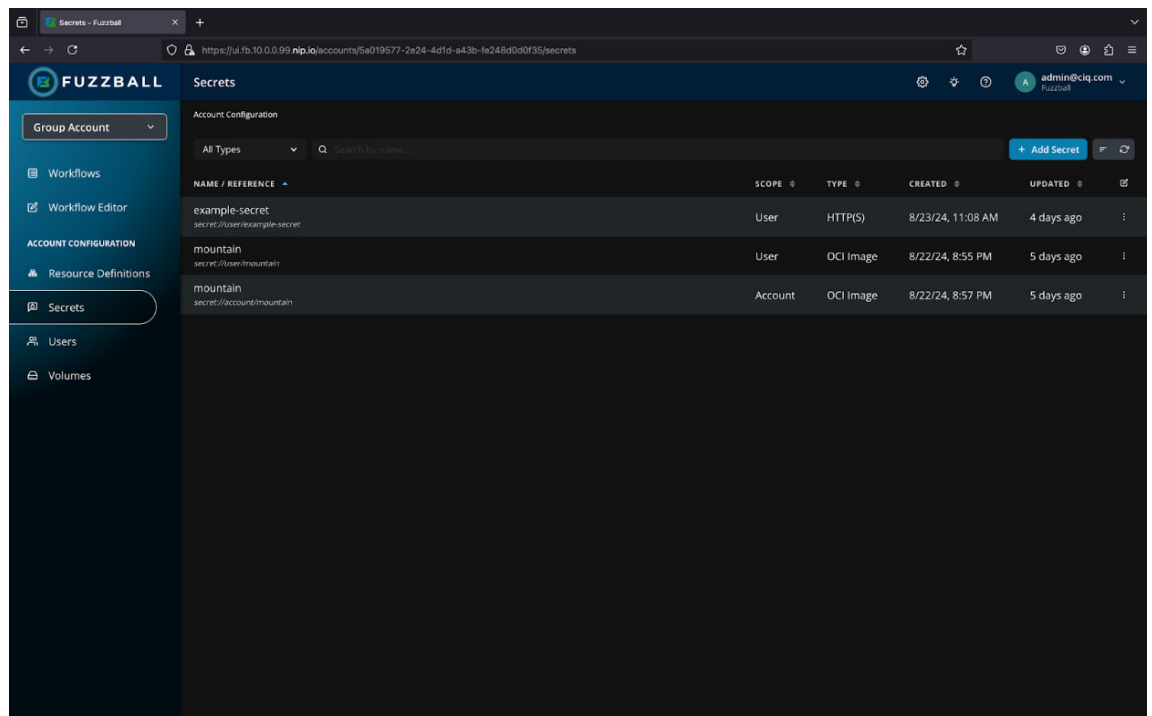
Once a given CSI driver is added to Fuzzball it can be used to define one or more **storage classes** which configure where and how data should be stored. **Storage volumes** can then be created based on those storage classes.

Storage volumes may be persistent or ephemeral. A persistent volume is created once and then may be used by later Fuzzball workflows to continue accessing the data that resides in it. Ephemeral volumes are created when a workflow starts and are discarded when a workflow ends. Ephemeral volumes are typically used for temporary storage, including to share data between multiple jobs in a workflow.

Ephemeral volumes often use Fuzzball's data ingress and egress support to bring in input data and to save results to permanent storage at the end of a workflow. Data may be transferred to and from an external S3 or HTTP(S) service.

Storage volumes may be scoped to the entire Fuzzball organization, for a specific Fuzzball account, or for a specific user account.

Secrets



Many HPC environments are deployed without any explicit secrets management facility. As such, credentials for external resources must either be handled by the system administrator or stored in plain-text files on the file system.

The Fuzzball **secrets** system securely stores sensitive data that is required for the proper execution of a Fuzzball workflow. This includes authentication credentials for S3 and HTTP(S) (for data ingress and egress) and OCI (for pulling container images from a registry); but secrets may also be defined for use as environment variables accessible directly within a job.

S3, HTTP(S), and OCI secrets are further scoped for use with one or more DNS domain names, ensuring that sensitive data is never exposed to the wrong service.

Deployment

Fuzzball is published using CIQ's Mountain content delivery service.

Fuzzball Orchestrate is a Kubernetes application, and deploys into a site's existing Kubernetes infrastructure (RKE2 preferred). Fuzzball orchestrate is deployed using [a custom Kubernetes operator](#) which provisions Orchestrate based on the configuration of a FuzzballOrchestrate custom resource definition (CRD). In this way, the configuration for the entire Orchestrate deployment may be managed and configured from a single document.

Fuzzball Substrate is deployed onto Enterprise Linux as a traditional RPM package. It may be installed using traditional tools, including DNF and Ascender²; but may also be deployed as part of an integrated compute node image using the Warewulf³ HPC cluster provisioning system.

And because Fuzzball handles user accounts and storage volumes within the application, there's nothing else to do! Once Fuzzball Substrate is running on compute nodes and connected to Fuzzball Orchestrate, available resources are automatically detected and made available for use. There's no additional configuration required to enumerate available resources, and no additional compute-node setup to prepare the node to receive user workloads.

Fuzzball surpasses challenges of traditional HPC

The patterns of traditional HPC represent the evolution of computing practices that have developed over more than 30 years. Fuzzball modernizes performance-intensive computing with a modern architecture that supports:

- Containerized workloads on industry-standard platforms
- A semantic, human-readable workflow definition language that supports portability between users and environments
- A user-friendly GUI for designing, editing, and executing HPC jobs
- Automated resource provisioning and management, including data ingress and egress and compliance logs
- Native integration with GPUs and both on-prem and cloud storage

²<https://ciq.com/products/ascender/>

³<https://warewulf.org/>

Fuzzball's approach ensures that both seasoned HPC professionals and newcomers can harness the power of performance computing without being bogged down by legacy expectations and requirements. Native support for containerized workloads, automated resource management, and seamless integration with modern computing environments guarantees that users can deploy, manage, and scale their HPC applications with ease and efficiency.

As businesses continue to face increasing demands for computational power, Fuzzball stands out as a solution that meets these needs head-on, offering a forward-looking platform that adapts to evolving technology practices.



Get Fuzzball for your on-prem HPC workload.
[Watch the Fuzzball overview demo video](#) to find out more.

Reach Us



CIQ.com



info@CIQ.com



800 220 5243

Connect with Us



linkedin.com/company/ctrliq



twitter.com/ctrliq



youtube.com/c/ctrliq

