BEGINNER TRAINING WITH iRODS 4.2

Part of the iRODS User Group Meeting

Utrecht, Netherlands

June 25, 2019

Contents

Introduction

Welcome!

This course is designed for those who are new to iRODS or who have limited experience with iRODS but want to learn more. Experience with the Unix command line and familiarity with the basic constructs of programming languages (e.g., variables, strings, loops) will be helpful to training participants.

In this course, you will implement an iRODS Zone (i.e., deployment) to satisfy the requirements of a hypothetical organization and set of users. The case study for this hypothetical organization will enable participants to become familiar with core iRODS functions. Through discussion and hands-on practice, you will:

- Gain an understanding of how iRODS is designed and works, and how it provides users with capabilities for virtualization, data discovery, workflow automation, and secure collaboration;
- Analyze user needs and determine how they can be operationalized in an iRODS deployment;
- Learn how to install and configure iRODS;
- Gain an understanding of virtualization in iRODS, including how to create a tree of resources to hold data;
- Gain an understanding of how data discovery can be improved with the use of metadata, and how iRODS handles metadata; and
- Gain an understanding of how workflows can be automated in iRODS through the use of rules and microservices.

You will leave this training with a foundational understanding of the overall technical structure and policy capabilities of iRODS, and the ability to execute core commands.

iRODS History

The iRODS story began in 1995 with a data management project known as Storage Resource Broker (SRB), led by Reagan Moore of the San Diego Supercomputer Center (SDSC). SRB is data grid middleware with a logical distributed file system, based on a client-server architecture that presents users with a single, global, logical namespace (i.e., file hierarchy). It was developed through the cooperative efforts of General Atomics, the Data Intensive Cyber Environments (DICE) group, and the SDSC at the University of California, San Diego (UCSD) with the support of the National Science Foundation (NSF).

The integrated Rule-Oriented Data System (iRODS), developed by the DICE group beginning in 2006, is SRB's successor. iRODS is based on SRB concepts, but was completely re-written to be fully open source and include a configurable rule engine.

In 2008, the DICE group expanded geographically, with some members accepting joint appointments in the School of Information & Library Science (SILS) and the Renaissance Computing Institute (RENCI) at the University of North Carolina at Chapel Hill (UNC). RENCI became progressively more involved in iRODS, culminating with the formation of a dedicated iRODS development team devoted to advancing iRODS to enterprise-grade quality. In 2013, RENCI founded the iRODS Consortium to further the mission and sustainability of iRODS technology.

About the iRODS Consortium

The iRODS Consortium is a group of organizations formally committed to iRODS' success through support for the development and release of iRODS-based data management and middleware technologies, promoting advances in iRODS, collaboration with iRODS developers and the iRODS open source community, attendance at iRODS events, and of course, membership dues.

The iRODS Consortium is operated at the University of North Carolina at Chapel Hill (UNC) by RENCI (Renaissance Computing Institute), a research institute of UNC. Consortium governance is provided through an Executive Board and Planning Committee. A Technology Working Group, composed of Consortium staff and satellite teams, contributes decision-making and development time to the project. Current Consortium members include RENCI, Bayer, The National Institute of Environmental Health Sciences (NIEHS), DataDirect Networks (DDN), Western Digital, Wellcome Sanger Institute, Utrecht University, MSC, University College London, Swedish National Infrastructure for Computing (SNIC), University of Groningen, SURF, Quantum, NetApp, Texas Advanced Computing Center (TACC), Cloudian, Maastricht University, University of Colorado Boulder, SUSE, and Aptiv. Membership is open to anyone interested in sustaining iRODS' success and participating in the iRODS community.

Consortium members receive a variety of benefits, including prioritized access to support, training, and consulting; and the opportunity to influence the developmental roadmap of future software releases.

Acknowledgements

In addition to the organizations and funding agencies listed below, the Consortium would like to specifically acknowledge Reagan Moore, Arcot Rajasekar, DICE, and RENCI.

Funded projects that supported the development of iRODS technology:

CyberCarpentry via DFC	NSF	11/1/2017 - 10/31/2020
SciDAS	NSF	2/15/2017 - 1/31/2020
DataBridge: Neuroscience	NSF	10/1/2016 - 9/30/2018
RADII	NSF	10/1/2014 - 9/30/2017
DataBridge: Social Science	NSF	11/1/2012 - 10/31/2015
HydroShare	NSF	7/1/2012 - 9/30/2021
EarthCube Layered Architecture	NSF	4/1/2012 - 3/31/2013
DFC Supplement for Extensible Hardware	NSF	9/1/2011 - 8/31/2015
DFC Supplement for Interoperability	NSF	9/1/2011 - 8/31/2015
DataNet Federation Consortium	NSF	9/1/2011 - 8/31/2016
SDCI Data Improvement	NSF	10/1/2010 - 9/30/2013
Subcontract: Temporal Dynamics of Learning Center	NSF	1/1/2010 - 12/31/2010
National Climatic Data Center	NOAA	10/1/2009 - 9/1/2010
NARA Transcontinental Persistent Archive Prototype	NSF	9/15/2009 - 9/30/2010
Subcontract: Temporal Dynamics of Learning Center	NSF	3/1/2009 - 12/31/2009
Transcontinental Persistent Archive Prototype	NSF	9/15/2008 - 8/31/2013
Petascale Cyberfacility for Seismic Community	NSF	4/1/2008 - 3/30/2010
Data Grids for Community Driven Applications	NSF	10/1/2007 - 9/30/2010
Joint Virtual Network Centric Warfare	DOD	11/1/2006 - 10/30/2007
Petascale Cyberfacility for Seismic Analysis	NSF	10/1/2006 - 9/30/2009
Digital Preservation Lifecycle Management	NSF	5/15/2005 - 4/30/2007
LLNL Scientific Data Management	LLNL	3/1/2005 - 12/31/2008
NARA Persistent Archives	NSF	10/1/2004 - 6/30/2008
Constraint-based Knowledge Systems	NSF	10/1/2004 - 9/30/2006
Biomedical Informatics Research Network (BIRN)	NIH	4/1/2004 - 4/1/2008
NDIIPP California Digital Library	LC	2/1/2004 - 1/31/2007
NASA Information Power Grid	NASA	10/1/2003 - 9/30/2004
Real-Time Data Aware System for Earth, Oceanographic, and Environmental Applications	NSF	9/15/2003 - 8/31/2007
Enabling the Science Environment for Ecological Knowledge (SEEK)	NSF	10/1/2002 - 9/30/2008
National Science Digital Library	NSF	10/1/2002 - 9/30/2006
NARA Persistent Archive	NSF	6/1/2002 - 5/31/2005
ROADNet	NSF	10/1/2001 - 9/30/2006
SCEC Community Modeling	NSF	10/1/2001 - 9/30/2006
Particle Physics Data Grid	DOE	8/15/2001 - 8/14/2004
Grid Physics Network	NSF	7/1/2000 - 6/30/2005
Knowledge Network for Biocomplexity (KNB)	NSF	9/15/1999 - 2/26/2005
Digital Library Initiative UCSB	NSF	9/1/1999 - 8/31/2004
Digital Library Initiative Stanford	NSF	9/1/1999 - 8/31/2004
Persistent Archive	NARA	9/1999 - 8/2000
Information Power Grid	NASA	10/1/1998 - 9/30/1999
Terascale Visualization	DOE	9/1/1998 - 8/31/2002
Persistent Archive	NARA	9/1998 - 8/1999
NPACI Data Management	NSF	10/1/1997 - 9/30/1999
DOE ASCI	DOE	10/1/1997 - 9/30/1999
Distributed Object Computation Testbed	DARPA/USPTO	8/1/1996 - 12/31/1999
Massive Data Analysis Systems	DARPA	9/1/1995 - 8/31/1996

Chapter 1

What is iRODS?

iRODS is open-source, data management middleware that enables users to:

- access, manage, and share data across any type or number of storage systems located anywhere, while maintaining redundancy and security, and
- exercise precise control over their data with extensible rules that ensure the data is archived, described, and replicated in accordance with their needs.

iRODS empowers users by supporting:

- **Virtualization**, which provides a one-stop shop for all data regardless of the heterogeneity of storage devices. Whether data is stored on a local hard drive, a remote Ceph cluster, or Amazon's S3 object store, iRODS' virtualization layer presents data resources in the classic files and folders format, within a single namespace.
- **Data Discovery** through the use of descriptive metadata,
- **Workflow Automation** through rules and microservices, and
- **Secure Collaboration** and data sharing between collaborating or distributed teams.

1.1 iRODS is Open Source

Open source software—such as iRODS—provides several benefits to users. First, because the source code is publicly available, the user community can monitor the entire development process. Second, developers within the user community can monitor and fix any errors in the code, extend the existing code, and contribute new code. For example, if a developer would like to add a custom authentication scheme, she can create a new plugin to handle this. Thus, iRODS code keeps improving and new functionality is continually added through community participation. Third, it allows Consortium members the opportunity to participate in the development of standards, software release roadmaps, and architectural plans, as well as provide oversight of development and testing efforts.

1.2 iRODS is Middleware

iRODS is a layer that sits above the file systems that contain data, and below domain-specific applications. Because iRODS has a plugin framework and is technology-agnostic, it provides insulation from vendor lock-in. System administrators can slide iRODS on top of an existing heterogeneous data infrastructure and construct a flexible data grid. As middleware, iRODS allows administrators to track and control access to the data under their care; and through Zone Reports (i.e., snapshots of an iRODS zone accessed via the izonereport iCommand), administrators can also monitor the status of the Zone (i.e., iRODS deployment).

1.3 An iRODS Zone

Each iRODS deployment—or Zone—is composed of an iRODS Metadata Catalog (iCAT) database, a Catalog Provider, and optional Catalog Consumers. The iCAT is a relational database that holds all the information about your data, users, and zone that the iRODS servers need to facilitate the management and sharing of your data. The iCAT contains the information about

- the zone for the purposes of sharing across zones,
- data and their metadata,
- the virtual file system,
- resource configuration, and
- user information.

Currently, PostgreSQL, MySQL, and Oracle are the officially supported database technologies that may be used to implement an iCAT database.

All iRODS servers in a Zone run the same core code and are peers. Each server may have its own set of policies, rules, and plugins. However, the Catalog Provider holds the connection to and communicates with the iCAT database. Consumer servers must communicate with the database by connecting through the Catalog Provider. A zone may have as many Consumer servers as needed. Using multiple Consumer servers can enhance the performance, security, and resilience of a Zone by providing redundancy, both within a single location and distributed geographically.

Data Objects and Metadata

In iRODS, the term Data Object refers to the logical representation of data that maps to one or more physical instances of the data at rest in storage resources, such as Amazon's S3. Data objects are organized into hierarchical Collections—the logical representations of physical containers, similar to directories or folders that are found in a file system. As with file system directories and folders, iRODS denotes levels of hierarchy with slashes (/) in the pathname. For example, the root collection of tempZone is written as /tempZone. Each subcollection is prefixed with /tempZone/. For example, /tempZone/home is a subcollection of /tempZone; and /tempZone/home/alice is a subcollection of /tempZone/home. The complete pathname of an iRODS data object includes the Zone (i.e., iRODS deployment) name and the full pathname within that zone, e.g., /tempZone/home/alice/sciproject/results.txt.

iRODS users can store descriptive information about data objects—or metadata—in the iCAT. Metadata improves search capabilities and therefore better enables data discovery. Users can search for data objects using metadata descriptors as search terms. This allows for browsing and serendipitous discovery, rather than relegating users to a targeted search for the file name, which they may or may not know. Both automatic, system-generated metadata and user-created metadata are supported in iRODS.

The Virtual File System and Resource Configuration

iRODS contains a virtual file system which maps logical directory paths stored in the iCAT to actual physical storage (e.g., Ceph cluster) that contains the logical data objects. Composable Resources allow you to manage storage and retrieval of data on storage devices. There are two types of composable resources: Coordinating and Storage. Coordinating resources actively make decisions about which physical device will receive or serve up a data object. Storage resources are the logical representations of—or pointers to—physical storage devices. All resources are composed of five parts: (a) the name you give the resource, (b) a host name (e.g., hostname.example.org), (c) a directory path to the exact location on the storage device (e.g., /full/path/to/storage), (d) the storage resource type (e.g., Amazon S3), and (e) a plugin-specific context string (e.g., the name of the file containing access credentials or any persistent information the plugin may require).

Secure Collaboration

With iRODS, organizations can share data, or federate, by simply adding a few bits of networking information to their iRODS configuration. Organizations are not required to coordinate the configuration of their respective iRODS zones. Each organization in a collaborative partnership retains autonomous control over its data collections, including maintaining security and data management policies distinct from fellow collaborators.

1.4 iRODS Rules

Your organization may have defined, formally or informally, policies and procedures for access control, backup, data migration, data preparation, metadata extraction, and more. These organizational policies can be implemented in iRODS through the use of rules, which can help you customize and automate related data management tasks.

Rules can be written in the iRODS' rule language, which uses many familiar programming constructs (e.g., loops, conditional statements), making it easy for your organization's developers to construct rules to satisfy your data needs. Rules can also be written in other supported languages if a rule engine plugin exists for that language. As of the time of publication, there are released rule engine plugins for the iRODS Rule Language, C++, and Python. There are experimental rule engine plugins for Javascript and Go.

Rules are executed based on conditions or, in iRODS parlance, Policy Enforcement Points (PEPs). Consider, for example, a rule to transfer ownership of data objects to the project manager when a user is deleted; the trigger—or PEP—is the deletion of the user. Similarly, rules could be written to extract metadata or pre-process data whenever a file is uploaded to a storage device. Or, upon access to particular data objects, a rule can create a log of the event, send an email notification to the project manager, or perform some other task you need to occur as a result of the data's access.

Rules are carried out by an iRODS rule engine plugin—a built-in interpreter for the rule language being used. The rule engine governs the sequence of data management actions in your iRODS zone. Rules are executed on the Catalog Provider or Consumer server that is handling the connection. Out of the box, the rule engine comes loaded with an array of basic actions (e.g., error reporting, permission checking) to get you up and running. As you begin to dig into iRODS, you can add rules of your own to tailor a data management program that works for you.

1.5 iRODS Plugins

We've already discussed two types of plugins: Composable Resources and Rule Engines; but there are many more. Plugins—like rules—allow for the customization of an iRODS installation. Plugins are used to implement core iRODS functions, such as authentication, communication over the Internet, communication with storage devices, and more. The use of plugins enables zone administrators to tailor iRODS to their needs, without having to recompile core code. Plugins also make it possible to upgrade small portions of iRODS without interfering with core functions.

Chapter 2

Case Study: stockphotosite.com (SPS)

Stockphotosite.com (SPS) is a fledgling startup that solicits and licenses stock photography. You are the Data Center Administrator for SPS. In that capacity, you have recruited a small network of freelance photographers who will upload images they capture to SPS, so they can be purchased and downloaded by subscribers.

Before SPS goes live, you will need to develop a data management strategy. To develop this strategy, you must take stock of what you have in-house and what your information needs are.

You have several computers and two 50 TB network-attached storage (NAS) arrays for supporting the site.

You need to support search and preservation of the images; so you need to gather information about each image, such as descriptions, photographer credits, copyright, resolution, camera settings, color or black and white, file format, etc.

You need to determine who can access the site and whether they are employees, photographers, or subscribers.

To ensure your site can grow over time, you will need to determine if and when data should be archived or deleted, and when expanding your storage configuration will be necessary. You can't afford downtime; you are expecting to have subscribers accessing the site 24/7 from around the world.

Disk failures occur, so you will want to keep at least two copies of the data, located on separate disks, at all times.

2.1 Addressing SPS's Needs

How would you address SPS's needs in the case study above?

Get together in pairs or small groups and discuss the questions below. The answers to some of the questions below cannot be found in the above scenario. So you'll have to use your imagination and flesh out the scenario some more.

Data

- Where is SPS' data located—one central location or distributed?
- What are the advantages and shortcomings of having the data in a central location; how about distributed? What do you think is the best course of action?
- Will SPS need multiple copies of data objects or is a single copy sufficient? What do you think is best?
- How much data does SPS currently have? Is this a stable quantity or could it increase over time? How quickly could SPS generate new data?
- What file formats will the data come in? Are these proprietary or open?
- Is any of the data proprietary or confidential?

Network

- What speed is the SPS network? How consistently is that speed maintained?
- What security is in place?

Resources

- Does SPS have a budget for ongoing software costs? For software engineers?
- What is SPS' stance on open source? How does this factor in to their decision-making process?
- What resources does SPS have in place to manage a data repository (e.g., technical staff, support options, site licenses, hardware)?

Organization

- What is SPS' organizational structure?
- Will you, the Data Center Administrator, have autonomous decision-making or will you need to get executive buy-in?
- Who is accountable for and affected by any decisions?

Users

- What kinds of users will need access to the data? What types of access or privileges will they need?
- How many users does SPS anticipate? Is this a stable quantity or could it increase over time?
- Where are the users located? What time zone? Are they located near the data or somewhere else?

Also consider the needs of specific classes of users:

- **SPS photographers** need to be able to document the photographs they upload. This includes text descriptions of the images that must be entered manually, but some descriptive information—metadata—does not require manual entry. Most digital cameras tag images with metadata stored as Exchangeable image file format (Exif) data. Exif data captures characteristics such as the date and time of the photograph, the make and model of the camera, the geographic coordinates of the subject, the lens aperture, the exposure time, and the focal length of the lens used to take the photograph.
- **Subscribers** need to be able to search for photos by their description, geographic location, image resolution, etc. They want easy, reliable access to the files they are entitled to; they are not concerned about where the files are located in the storage array.
- **You, as the Data Center Administrator**, need to be able to verify that two copies of each image are maintained without interfering with user access. You need to implement archiving and retention policies, and you need to add new storage as existing resources fill up.

The questions above can serve as an inventory-taking template for your own personal situation. Just substitute "SPS" with your organization.

2.2 Planning an iRODS Deployment for SPS

As SPS' Data Center Administrator, you need to consider the questions above and decide how these requirements can be operationalized in an iRODS deployment.

Get together in pairs or small groups and discuss your requirements. You may need to scan later chapters of the workbook.

Data, Network, Resource, and Organizational Needs

Start by considering the requirements you identified when reviewing the questions related to Data, Network, Resources, and Organizational needs. For example, if SPS data is distributed across storage devices, you may need multiple resource servers and a robust tree of composable resources. If you think SPS will need multiple copies of data objects, you may want to look into iRODS' capabilities for replication: take a look at the Coordinating Resources table in the Virtualization chapter of this workbook. If you are pro-open-source but your organization is unfamiliar with the concept, how can you educate them on the benefits to ensure adoption of iRODS?

Users Needs

User needs are very important. If the data management system won't support user needs, it will not be useful to them and the system may not be adopted. What requirements did you identify when considering the User questions above? For example, if you anticipate that SPS will cater to large numbers of subscribers from around the globe, you may want resource servers to be located in the countries with the largest groups of subscribers. Will the photographers want to be able to annotate their photos with descriptive information? What other kinds of user needs do you think will be important to attend to?

Chapter 3

Roles

Throughout the training, you will be assuming different user roles to interact with the Linux VM and the iRODS software. The table below lists the different roles you will assume and explains their purpose.

Role	Linux, iRODS, or Postgres?	Role Definition
ubuntu	Linux	This is the user account on the Linux VM. You will be `ubuntu` throughout most of the day because you will always be using the Linux VM.
postgres	Linux	This is the service account for the Postgres database management system. We will use this account to create the iCAT.
irods	Postgres	This is the database user account which will own the iCAT.
irods	Linux	This is the Linux service account that is created by default when you install iRODS. It runs the iRODS software and owns all physical data objects.
rods	iRODS	This is the default administrator account—rodsadmin—in iRODS. It has permissions to add users to iRODS, set up resources, etc.
alice	iRODS	A regular user account—rodsuser—in iRODS. A photographer at SPS.
bobby	iRODS	A regular user account—rodsuser—in iRODS. A photographer at SPS.

Chapter 4

Installing iRODS

Before installing iRODS, we first need to install and configure a database.

4.1 The iCAT Database

iRODS stores most of its information (e.g. user names, file names and locations, metadata) in the iCAT. iRODS assumes this database is created and managed by a third party. Therefore, before installing iRODS, we have to create and configure the database iRODS will be using.

For this training, we will be using Ubuntu for our operating system and PostgreSQL for our iRODS database. First let's update Ubuntu's apt repository.

```
$ sudo apt-get update
```

Then let's install the PostgreSQL server software.

```
$ sudo apt-get -y install postgresql
```

Next, we will switch user to the Linux user account—postgres—that controls the PostgreSQL server software so that we can create the iCAT database:

```
$ sudo su - postgres
```

Start the PostgreSQL command console:

```
$ psql
```

Now we are in PostgreSQL, so we will switch to database query language.

Let's create the database to be used by iRODS:

```
> CREATE DATABASE "ICAT";
```

Create the PostgreSQL user account to be used by iRODS:

```
  > CREATE USER irods WITH PASSWORD 'testpassword';
```

Give the iRODS PostgreSQL user account permission to use the database:

```
> GRANT ALL PRIVILEGES ON DATABASE "ICAT" to irods;
```

Log out of the PostgreSQL command console:

```
> \q
```

Log out of the Linux user account—postgres—that controls the PostgreSQL server software:

```
$ exit
```

You are now once again `ubuntu`.

4.2 Installing iRODS Software Packages

iRODS is split into a series of packages that declare dependencies on one another:

- the core server software
- the database plugin specific to the type of database used (PostgreSQL in our case)
- the iCommands
- the runtime (shared libraries used by other iRODS packages)

iRODS software can be installed via the native operating system's package manager.

To install the RENCI repository that hosts the iRODS software, execute the **APT** instructions located at `https://packages.irods.org`:

```
$ wget -qO - https://packages.irods.org/irods-signing-key.asc | \
    sudo apt-key add -
$ echo "deb [arch=amd64] https://packages.irods.org/apt/ \
    $(lsb_release -sc) main" | sudo tee \
    /etc/apt/sources.list.d/renci-irods.list
$ sudo apt-get update
```

To download the core server software execute:

```
$ sudo apt-get -y install irods-server irods-database-plugin-postgres
```

Then you will be presented with, among other things, this set of messages:

```
iRODS Postgres Database Plugin installation was successful.

To configure this plugin, the following prerequisites need to be met:
 - an existing database user (to be used by the iRODS server)
 - an existing database (to be used as the iCAT catalog)
 - permissions for existing user on existing database

Then run the following setup script:
  sudo python /var/lib/irods/scripts/setup_irods.py
```

The final installation step is running the setup script:

```
$ sudo python /var/lib/irods/scripts/setup_irods.py
```

The `setup_irods.py` script will ask for information in five sections.

Default values are shown in grey boxes. For this installation, we will use the default values.

1. Service Account	
• Service Account Name	irods The Linux account that will run the iRODS server software. The account will be created if it does not already exist.
• Service Account Group	irods The primary group of the Linux account that will run the iRODS server software.
• Catalog Service Role	1 Provider or Consumer, determines whether this server holds a connection to the metadata catalog (database).
2. Database Connection (if installing a 'Provider')	
• ODBC Driver	1 The driver on the server used to talk to the ODBC database layer.
• Database Server's Hostname or IP	localhost We are connecting to a local database.
• Database Server's Port	5432 The database server listens for notifications from other applications on this port. The default value is correct for PostgreSQL installations.
• Database Name	ICAT This is the name of the database that we created in PostgreSQL during the iCAT database installation.
• Database User	irods This must match the irods Linux account name to authenticate into Postgres without changing Postgres settings.
• Database Password	Enter `testpassword` . This is the same password we set during the iCAT database installation.
• Stored Passwords Salt	This obfuscates the passwords stored in the database, making your installation unique. For production use, don't leave this empty.
3. iRODS Server Options	
• Zone Name	tempZone The name of the iRODS zone.
• Zone Port	1247 The main iRODS port.
• Parallel Port Range (Begin)	20000 The beginning of the port range used when transferring large files.
• Parallel Port Range (End)	20199 The end of the port range used when transferring large files.
• Control Plane Port	1248 The port used for the control plane. The control plane receives status updates from all servers, and issues commands to servers to pause, resume, shutdown, etc.
• Schema Validation Base URI	file:///var/lib/irods/configuration_schemas The location of the schema files used to validate the server's configuration files.
• iRODS Administrator Username	rods The name of the iRODS administration account that will be created during setup.
4. Keys and Passwords	
• Zone Key	A secret key used in server-to-server communication.
• Negotiation Key	A secret key used in server-to-server communication. Must be exactly 32 bytes (characters).
• Control Plane Key	A secret key shared by all servers. Must be exactly 32 bytes (characters).
• iRODS Administrator Password	For the purposes of this class, use `rods`. In the future, however, you will want to use more complex passwords.
5. Default Vault Information	
• Vault Directory	/var/lib/irods/Vault The Vault (i.e., storage) location of the default unix-filesystem resource created during installation.

Once `setup_irods.py` has received all of its input, it will complete the setup.

A successful setup will end with the following text:

```
+---------------------+
| Attempting test put |
+---------------------+

Putting the test file into iRODS...
Getting the test file from iRODS...
Removing the test file from iRODS...
Success.

+--------------------------------+
| iRODS is installed and running |
+--------------------------------+
```

Chapter 5

Using iCommands

iCommands are Unix utilities that give iRODS users a command-line interface to operate on data in the iRODS system. iCommands provide client-side communication with iRODS servers to provide administrative, data management, and metadata management functions. There are over 50 iCommands; and in this course, we will cover most of the basic iCommands.

All iCommands accept command line options (e.g., -a for all, -e for echo, -h for help) that extend the command's capabilities. However, a specific iCommand accepts only a subset of these options. The options that an iCommand accepts are listed in its help entry.

To get help on a specific iCommand (such as `ils` for listing the contents of a collection), you may

- visit this webpage: `https://docs.irods.org/4.2.6/icommands/user/`
- use the -h option with the command (e.g., `ils -h`), or
- use the `ihelp` command as the argument with the command you'd like to learn more about (e.g., `ihelp ils`).

5.1 Administrative Operations

In this workshop, you are interacting with iRODS through a Linux shell session. The session needs many settings and other details to determine iRODS behavior and access to iRODS resources. One way the shell manages these settings and details is through an area it maintains called the environment. The shell builds the environment every time it starts a session by accessing the settings and details that are contained in an environment file.

So to connect to an iRODS server using the rods administrator account (i.e. `rodsadmin`), we will need to execute the `iinit` command which will create the iRODS environment file `irods_environment.json`, in the `.irods` subdirectory of your Linux home directory (e.g., `/home/ubuntu/.irods/irods_environment.json`).

You will need to have the following information handy when you run `iinit`:

- the hostname of the iRODS server (either a catalog provider or catalog consumer) you wish to log into: `X.X.X.X`
- the network port number of the iRODS server: `1247`
- the name and password of the iRODS user: `rods`
- the name of the iRODS zone: `tempZone`

Execute `iinit`.

On the same line, after the colon, enter: `X.X.X.X`

```
$ iinit
One or more fields in your iRODS environment file (irods_environment.json) are missing; please enter them.
Enter the host name (DNS) of the server to connect to:
```

On the same line, after the colon, enter: `1247`

```
Enter the port number:
```

On the same line, after the colon, enter: `rods`

```
Enter your irods user name:
```

On the same line, after the colon, enter: `tempZone`

```
Enter your irods zone:
```

Remember we set the rods password as `rods`

```
Those values will be added to your environment file (for use by
other iCommands) if the login succeeds.

Enter your current iRODS password:
```

The `iinit` command will cache your credentials in the file `~/.irods/.irodsA`.

Now let's create accounts for two SPS photographers: Alice Jones and Bobby Smith. To perform administrative functions such as adding, modifying, and removing users and storage resources, we will use the iadmin command.

Use `iadmin` with the make user (`mkuser`) argument to create a user account (rodsuser) for Alice and assign her the password `passWORD`. We are going to use lowercase for Alice's username. Then we will use `moduser` to change properties of a user account (e.g., the password).

```
$ iadmin mkuser alice rodsuser
$ iadmin moduser alice password passWORD
```

If you wanted Alice to be an iRODS administrator, you would use `rodsadmin` in place of `rodsuser`. An administrator with a rodsadmin account—such as rods—has permission to run `iadmin` and perform other administrative activities. For now, leave Alice as `rodsuser`.

Let's create one more regular user account for Bobby with the same password we gave Alice.

```
$ iadmin mkuser bobby rodsuser
$ iadmin moduser bobby password passWORD
```

Let's create a storage resource so that Alice can try out iCommands in the next subsection. When we installed iRODS, the setup script created an initial iRODS storage resource, `demoResc`.

Please Note - demoResc is not for production use, but we will use it for training purposes.

Let's use iadmin to create a second resource, `newResc`, of the type unixfilesystem, on the local host, and mounted at `/var/lib/irods/new_vault`. This will be a single line of text on your screen.

```
$ iadmin mkresc newResc unixfilesystem `hostname`:/var/lib/irods/new_vault
```

The iadmin command can also be used to remove a user (using rmuser as the argument) or remove a resource (using rmresc as the argument). Use iadmin -h to learn more.

Now that we're done with administrative commands, let's log out of iRODS.

```
$ iexit full
```

To log in as Alice in the next section, we'll need to throw away the environment file for `rods`. To do this, execute:

```
$ rm ~/.irods/irods_environment.json
```

5.2 Logging In with Alice

Now, using the `iinit` command, let's log in as Alice and change her password.

Execute `iinit`.

On the same line, after the colon, enter: `X.X.X.X`

```
$ iinit
One or more fields in your iRODS environment file (irods_environment.json) are missing; please enter them.
Enter the host name (DNS) of the server to connect to:
```

On the same line, after the colon, enter: `1247`

```
Enter the port number:
```

On the same line, after the colon, enter: `alice`

```
Enter your irods user name:
```

On the same line, after the colon, enter: `tempZone`

```
Enter your irods zone:
```

Remember we set Alice's password as `passWORD`. We must log in first before we can change it.

```
Those values will be added to your environment file (for use by
other iCommands) if the login succeeds.

Enter your current iRODS password:
```

Let's change Alice's password using the `ipasswd` command. Remember Alice's current iRODS password was `passWORD`.

```
$ ipasswd
Enter your current iRODS password:
Enter your new iRODS password:
Reenter your new iRODS password:
```

Enter a new password for Alice. For the purposes of this training, use `alicepass`.

5.3 Basic Navigation

Unix commands such as cd, ls, and pwd are available in iRODS as iCommands. To identify the current working collection use the `ipwd` command. The current working collection is the default location for data to be read or written.

```
$ ipwd
/tempZone/home/alice
```

Now, let's change to another collection using the `icd` command. To change the collection to `/tempZone/home/public`, you would use `icd` with an absolute path:

```
$ icd /tempZone/home/public
```

Or you could use a relative path:

```
$ icd ../public
```

To list the data objects and subcollections stored in a collection, we could use the `ils` (meaning list) command. If executed with no arguments, it will list the objects in the present collection.

```
$ ils
/tempZone/home/public:
```

You can see that public is empty. There are no data objects or subcollections in public. To revisit Alice's home collection you can use use the `icd` command by itself or followed by Alice's home collection.

```
$ icd
```

Or

```
$ icd ../alice
```

5.4 Working with Data Objects

For this training, we will be using a set of images. To prepare your home directory for the following exercises, do the following:

```
$ wget https://github.com/irods/irods_training/raw/ugm2019/beginner/training_jpgs.zip
$ sudo apt-get -y install unzip
$ unzip training_jpgs.zip
```

Suppose Alice would like to upload her photos from a local directory to an iRODS collection as data objects. For this, use the `iput` command:

```
$ iput -r /home/ubuntu/training_jpgs
```

If Alice wanted to copy `lemur.jpg` from an iRODS collection to her local directory, she would use `iget`:

```
$ iget /tempZone/home/alice/training_jpgs/lemur.jpg
```

Using the `-` command line option at the end will print the contents to stdout but will not create a local copy.

```
$ iget /tempZone/home/alice/training_jpgs/sources.txt -
```

Let's give Bobby access to `mouse.jpg`, but first we need to give Bobby read access to Alice's `training_jpgs` collection that contains `mouse.jpg`. Let's start by reviewing what permissions are already in place with the ils command followed by the `-A` and `-r` options. The `-A` option will show you the data objects' and collections' access control lists (ACLs) which define who owns or who has read/write permissions to the data and collections. The `-r` option here applies the `ils` command to the target and all its subcollections.

```
$ ils -A -r
/tempZone/home/alice:
        ACL - alice#tempZone:own
        Inheritance - Disabled
 C- /tempZone/home/alice/training_jpgs
/tempZone/home/alice/training_jpgs:
        ACL - alice#tempZone:own
        Inheritance - Disabled
 beans.jpg
        ACL - alice#tempZone:own
 coffee.jpg
        ACL - alice#tempZone:own
 eggs.jpg
        ACL - alice#tempZone:own
 grapes.jpg
       ACL - alice#tempZone:own
 lemur.jpg
       ACL - alice#tempZone:own
 mouse.jpg
       ACL - alice#tempZone:own
 peanuts.jpg
       ACL - alice#tempZone:own
 platter.jpg
       ACL - alice#tempZone:own
 scooter.jpg
       ACL - alice#tempZone:own
 seal.jpg
       ACL - alice#tempZone:own
 sources.txt
       ACL - alice#tempZone:own
 waffle.jpg
       ACL - alice#tempZone:own
```

Bobby does not have access to `training_jpgs`. Alice owns the `/tempZone/home/alice/training_jpgs` collection. Her ownership permission allows her to grant permissions to others. So as Alice, let's give Bobby write permissions using the `ichmod` command followed by the `-r` command line option to apply the new permission recursively.

```
$ ichmod -r write bobby training_jpgs
```

Write permissions in iRODS include read permissions; however, you can grant read-only permissions by using read in place of write in the above command.

Let's make sure that it worked. We'll use `ils` again to review the permissions:

```
$ ils -A -r
/tempZone/home/alice:
       ACL - alice#tempZone:own
       Inheritance - Disabled
 C- /tempZone/home/alice/training_jpgs
/tempZone/home/alice/training_jpgs:
       ACL - alice#tempZone:own   bobby#tempZone:modify object
       Inheritance - Disabled
 beans.jpg
       ACL - bobby#tempZone:modify object   alice#tempZone:own
 coffee.jpg
       ACL - bobby#tempZone:modify object   alice#tempZone:own
 eggs.jpg
       ACL - bobby#tempZone:modify object   alice#tempZone:own
 grapes.jpg
       ACL - bobby#tempZone:modify object   alice#tempZone:own
 lemur.jpg
       ACL - bobby#tempZone:modify object   alice#tempZone:own
 mouse.jpg
       ACL - bobby#tempZone:modify object   alice#tempZone:own
```

```
peanuts.jpg
      ACL - bobby#tempZone:modify object   alice#tempZone:own
platter.jpg
      ACL - bobby#tempZone:modify object   alice#tempZone:own
scooter.jpg
      ACL - bobby#tempZone:modify object   alice#tempZone:own
seal.jpg
      ACL - bobby#tempZone:modify object   alice#tempZone:own
sources.txt
      ACL - bobby#tempZone:modify object   alice#tempZone:own
waffle.jpg
      ACL - bobby#tempZone:modify object   alice#tempZone:own
```

If we want to remove a data object from a collection, the `irm` command will do that. By default, `irm` moves the data object to a separate trash collection at `/tempZone/trash`.

```
$ irm /tempZone/home/alice/training_jpgs/peanuts.jpg
```

To empty the trash, we could use `irmtrash`. Using the `-f` command line option with `irm` will remove the data object permanently, instead of moving it to the trash. It cannot be recovered if you use `-f`.

Suppose you did not intend to throw `peanuts.jpg` away. Thankfully you did not use the `-f` option, so you can use the `imv` command to recover the data object from the trash and move it back to Alice's collection.

```
$ imv /tempZone/trash/home/alice/training_jpgs/peanuts.jpg /tempZone/home/alice/training_jpgs
```

Now let's verify that `peanuts.jpg` was moved back to Alice's collection:

```
$ ils /tempZone/home/alice/training_jpgs/peanuts.jpg
```

Now, let's create a Replica (i.e., an identical, physical copy of a data object) using the `irepl` command. First, let's use ils to take a look at what's inside `/tempZone/home/alice/training_jpgs` and we'll use the `-L` option to determine if there are any replicas already in existence. Using the `-L` command line option (meaning very long) with `ils` will show you where data objects are physically stored and if replicas exist.

```
$ ils -L /tempZone/home/alice/training_jpgs
/tempZone/home/alice/training_jpgs:
  alice             0 demoResc      1128069 2019-04-07.14:16 & beans.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/beans.jpg
  alice             0 demoResc       479299 2019-04-07.14:16 & coffee.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/coffee.jpg
  alice             0 demoResc       912548 2019-04-07.14:16 & eggs.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/eggs.jpg
  alice             0 demoResc       669306 2019-04-07.14:16 & grapes.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/grapes.jpg
  alice             0 demoResc      1312007 2019-04-07.14:16 & lemur.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/lemur.jpg
  alice             0 demoResc       392585 2019-04-07.14:16 & mouse.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/mouse.jpg
  alice             0 demoResc      1413230 2019-04-07.14:16 & peanuts.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/peanuts.jpg
  alice             0 demoResc      2555592 2019-04-07.14:16 & platter.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/platter.jpg
  alice             0 demoResc      1822077 2019-04-07.14:16 & scooter.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/scooter.jpg
  alice             0 demoResc       362833 2019-04-07.14:16 & seal.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/seal.jpg
  alice             0 demoResc          371 2019-04-07.14:16 & sources.txt
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/sources.txt
  alice             0 demoResc      1153142 2019-04-07.14:16 & waffle.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/waffle.jpg
```

We can see that there is a single replica of each data object (they are all on demoResc and are replica zero). Each data object refers to only one replica right now.

Let's replicate `peanuts.jpg` to `newResc` using the `irepl` command:

```
$ irepl -R newResc training_jpgs/peanuts.jpg
```

Now, let's verify that the replica was created (now numbered `1` on `newResc`):

```
$ ils -L training_jpgs
/tempZone/home/alice/training_jpgs:
  alice             0 demoResc      1128069 2019-04-07.14:16 & beans.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/beans.jpg
  alice             0 demoResc       479299 2019-04-07.14:16 & coffee.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/coffee.jpg
  alice             0 demoResc       912548 2019-04-07.14:16 & eggs.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/eggs.jpg
  alice             0 demoResc       669306 2019-04-07.14:16 & grapes.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/grapes.jpg
  alice             0 demoResc      1312007 2019-04-07.14:16 & lemur.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/lemur.jpg
  alice             0 demoResc       392585 2019-04-07.14:16 & mouse.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/mouse.jpg
  alice             0 demoResc      1413230 2019-04-07.14:16 & peanuts.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/peanuts.jpg
  alice             1 newResc       1413230 2019-04-07.14:37 & peanuts.jpg
        newResc     generic    /var/lib/irods/new_vault/home/alice/training_jpgs/peanuts.jpg
  alice             0 demoResc      2555592 2019-04-07.14:16 & platter.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/platter.jpg
  alice             0 demoResc      1822077 2019-04-07.14:16 & scooter.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/scooter.jpg
  alice             0 demoResc       362833 2019-04-07.14:16 & seal.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/seal.jpg
  alice             0 demoResc          371 2019-04-07.14:16 & sources.txt
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/sources.txt
```

Suppose you wish to remove a replica. The `itrim` command is suited to the job. Let's remove the replica of `peanuts.jpg` from `demoResc`. If we add the `-N` command line option, we can specify the number of replicas to keep. To keep only 1 replica, follow `-N` with number 1. To keep 3, follow it with 3, and so on. If you do not specify a number, iRODS will trim replicas down to 2 by default.

Currently, we have two replicas of `peanuts.jpg`—one on `demoResc` and one on `newResc`. To specify that we wish the replica to be trimmed from `demoResc`, we will need to use the `-S` option followed by `demoResc`. The `-S` option specifies the resources of the replica to be trimmed.

```
$ itrim -N 1 -S demoResc training_jpgs/peanuts.jpg
```

Now let's verify that the replica of `peanuts.jpg` on `demoResc` has been successfully trimmed by using `ils -L` again.

```
$ ils -L training_jpgs
/tempZone/home/alice/training_jpgs:
 alice              0 demoResc      1128069 2019-04-07.14:16 & beans.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/beans.jpg
 alice              0 demoResc       479299 2019-04-07.14:16 & coffee.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/coffee.jpg
 alice              0 demoResc       912548 2019-04-07.14:16 & eggs.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/eggs.jpg
 alice              0 demoResc       669306 2019-04-07.14:16 & grapes.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/grapes.jpg
 alice              0 demoResc      1312007 2019-04-07.14:16 & lemur.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/lemur.jpg
 alice              0 demoResc       392585 2019-04-07.14:16 & mouse.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/mouse.jpg
alice               1 newResc       1413230 2019-04-07.14:37 & peanuts.jpg
        newResc     generic    /var/lib/irods/new_vault/home/alice/training_jpgs/peanuts.jpg
 alice              0 demoResc      2555592 2019-04-07.14:16 & platter.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/platter.jpg
 alice              0 demoResc      1822077 2019-04-07.14:16 & scooter.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/scooter.jpg
 alice              0 demoResc       362833 2019-04-07.14:16 & seal.jpg
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/seal.jpg
 alice              0 demoResc          371 2019-04-07.14:16 & sources.txt
        demoResc    generic    /var/lib/irods/Vault/home/alice/training_jpgs/sources.txt
```

We have trimmed the replica only from `demoResc`. The data object `peanuts.jpg` now only appears on `newResc`.

5.5 Making Collections

Alice might want to organize her photos into different collections—perhaps putting black and white photos in one collection and color in another. You can make collections with the `imkdir` command followed by the collection name. The collection name you specify can be relative to your current working collection:

```
$ imkdir bw_photos
```

or absolute, as shown below:

```
$ imkdir /tempZone/home/alice/bw_photos
```

To remove a collection, use the `irm` command with the `-r` command line option, followed by the collection you wish to remove. The `-r` command line option will apply the remove command recursively.

```
$ irm -r bw_photos
```

Or by referencing the full path:

```
$ irm -r /tempZone/home/alice/bw_photos
```

5.6 Wrapping Up

We have covered several fundamental iCommands, but there are dozens more that cover copying data between collections, moving data between resources, computing checksums, etc.

You can learn more about any iCommand using the `-h` command line option, or by visiting `https://docs.irods.org/4.2.6/icommands/user/`.

Chapter 6

Virtualization

In computer science, virtualization is the process of providing one abstract interface—or virtual access point—through which multiple services or entities communicate. These services or entities can include servers, storage devices, networks, and operating systems.

iRODS has a modular architecture with 7 pluggable interfaces that allow different services (i.e., iRODS plugins) to communicate with iRODS core code. Because these plugins are separate from core code, new functionality can be added without having to edit or recompile core code.

Through these interfaces, chiefly composable resources, iRODS users can take advantage of storage virtualization. iRODS users are provided with a uniform interface that supports the implementation of consistent data management policies and practices regardless of different types or numbers of storage. For example, data could be stored on Amazon S3, a Unix file system, and Web Object Scaler (DataDirect Network's block storage appliance), yet data retrieval, sharing, and replication could be handled in the same way regardless of device differences.

Interface	Plugins
Authentication	Native (iRODS password) Pluggable Authentication Module (PAM) LDAP via PAM Kerberos* Grid Security Infrastructure (GSI)*
Network	Transmission Control Protocol (TCP) Secure Socket Layer (SSL)
Database	Oracle* PostgreSQL* MySQL* CockroachDB (unreleased)*
Rule Engine	iRODS Rule Language Python* C++ (various)* Javascript (unreleased)* Go (unreleased)*
API	An avenue through which new functionality can be added to an iRODS deployment.
Microservices	There are over 350 available, covering a variety of functions.
Composable Resources	There are two kinds of Composable Resources in iRODS: Coordinating and Storage. Both are discussed in more detail in the next section.

* Installed separately. All other plugins listed are installed by default.

6.1 Resource Composition

Recall that Composable Resources are plugins that allow you to create rich decision trees for managing storage and retrieval of data, independent of storage device. There are two types of Composable Resources: Coordinating and Storage. One way to think about composable resources is to consider coordinating resources as the branch nodes of your decision tree and storage resources as the leaves.

Coordinating Resources

A Coordinating Resource manages the flow of data to and from Storage Resources. In the iCAT database, they are composed of three parts: (a) the name you give the resource, (b) the resource type (e.g., Replication), and (c) a plugin-specific context string (e.g., in the case of a Passthru resource, the read and write weights—see below).

Replication	A replication resource keeps its children in sync with identical copies—or replicas—of data objects.
Load Balanced	A load balanced resource attempts to balance storage and retrieval across children to avoid taxing the servers (e.g., available space, CPU usage, network traffic).
Deferred	A deferred resource has no implicit policy and will honor the voting of its children in determining priority for both puts and gets.
Compound	A compound resource manages two children, in the roles of Cache and Archive. The cache resource provides a standard UNIX file system interface (i.e., POSIX) to an archive that may not natively support this type of access.
Random	With a multitude of resources, a random resource can be quite successful in distributing replicated data objects evenly across disparate storage devices.
Passthru	The passthru resource allows administrators to prioritize one or more Storage Resources (and thus storage devices) over others with a set of weighting scores for reads and writes.

Storage Resources

Storage Resources are the logical representations of—or pointers to—physical storage devices. They are composed of five parts: (a) the name you give the resource, (b) a host name (e.g., host.example.org), (c) a directory path—or Vault—to the exact location on the storage device (e.g., `/full/path/to/storage`), (d) the resource type (e.g., Amazon S3), and (e) a plugin-specific context string (e.g., the name of the file containing access credentials or any persistent information the plugin may require).

Unix File System	This type of storage resource communicates with a storage device through the standard POSIX interface
S3*	Amazon's or any other S3-compatible storage service
Web Object Scaler (WOS)*	DataDirect Networks' block storage appliance
Ceph RADOS (unreleased)*	Designed for efficient cacheless access to a Ceph RADOS block storage cluster
Universal Mass Storage	For use with Compound resources, such as tape-based archives

* Installed separately. All other plugins listed are installed by default.

6.2 How Composable Resources Communicate

When a request for data is made, each storage resource communicates with its parent coordinating resource about whether it is able to provide the requested data. The coordinating resource then decides which particular storage resource is the best option for serving up the data. This is determined based on the nature of the coordinating resource.

For example, the Replication Coordinating Resource weights a vote from a storage resource more heavily if it possesses a local copy. The replication coordinating resource is designed to honor "locality of reference," and then point the client to the closest data.

6.3 Building a Tree

Recall that we created a unixfilesystem storage resource called `newResc` for Alice to play with. Now we're going to set up automatic replication between `newResc` and two other resources, which will be named `storageResc1` and `storageResc2`.

To set up automatic replication, we will need to use the `iadmin` command, so we will need to `iexit full` out of Alice's account and log in as `rods`, the administrator account.

```
$ iexit full
```

To log in as rods, we'll need to throw away Alice's environment file. To do this, execute:

```
$ rm ~/.irods/irods_environment.json
```

Now let's log in to the rods account:

Execute `iinit`.

On the same line, after the colon, enter: `X.X.X.X`

```
$ iinit
One or more fields in your iRODS environment file (irods_environment.json) are missing; please enter them.
Enter the host name (DNS) of the server to connect to:
```

On the same line, after the colon, enter: `1247`

```
Enter the port number:
```

On the same line, after the colon, enter: `rods`

```
Enter your irods user name:
```

On the same line, after the colon, enter: `tempZone`

```
Enter your irods zone:
```

Remember we set the rods password as `rods`.

```
Those values will be added to your environment file (for use by
other iCommands) if the login succeeds.

Enter your current iRODS password:
```

Now we are ready to create some resources. First, we need to create `storageResc1` and `storageResc2`, which will also be unixfilesystem resources.

```
$ iadmin mkresc storageResc1 unixfilesystem `hostname`:/var/lib/irods/storageVault1
$ iadmin mkresc storageResc2 unixfilesystem `hostname`:/var/lib/irods/storageVault2
```

Now we will create a replication coordinating resource named `replResc`.

```
$ iadmin mkresc replResc replication
```

Once `replResc` exists, we need to connect `newResc`, `storageResc1`, and `storageResc2` as children of `replResc`.

```
$ iadmin addchildtoresc replResc newResc
$ iadmin addchildtoresc replResc storageResc1
$ iadmin addchildtoresc replResc storageResc2
```

When writing data (usually via an `iput`), the default replication coordinating resource populates one of the children resources with the data object, and then replicates the data object to the remaining children.

6.4 Seeing the Tree

The composable resources can always be visualized with the ilsresc command.

```
$ ilsresc --ascii
demoResc
replResc:replication
|-- newResc
|-- storageResc1
`-- storageResc2
```

6.5 Adding New Storage

After we have created the resource tree, we need to tell the replication resource to make sure all of its children have copies of all of the same files. The rebalance command accomplishes this process on a replication resource. This may take some time, but can be stopped and restarted safely.

```
$ iadmin modresc replResc rebalance
```

6.6 Decommissioning Storage

As our needs change, we may wish to decommission storage resources to use for other purposes. The following commands will remove `storageResc1` from the resource tree so it can be repurposed.

```
$ iadmin rmchildfromresc replResc storageResc1
$ itrim -M -r -S storageResc1 /tempZone
$ iadmin rmresc storageResc1
```

The first command above severs the parent-child relationship between `replResc` and `storageResc1`, making `storageResc1` a free-standing storage resource (but still fully functional and available to store and retrieve data objects).

Next, all the replicas on `storageResc1` (`-S storageResc1`) are recursively (-r) trimmed. This is done in admin mode (-M) so other users' data objects are trimmed as well.

Once there is no data stored on `storageResc1`, it can be safely removed as a storage resource under iRODS management. The disk can be retired with no effect on the running system.

Chapter 7

Data Discovery

iRODS employs a metadata catalog—iCAT—that permits users and administrators to access and contribute descriptive information about their data. This descriptive information—or metadata—improves search and therefore better enables data discovery. Users can search for data objects using metadata descriptors as search terms. This allows for browsing and serendipitous discovery, rather than relegating users to a targeted search for a file name they may not know.

Both automatic, system-generated metadata and user-created metadata are supported in iRODS. For example, iRODS can automatically derive the data object's creation date, modification date, the last date it was accessed, etc.; and an individual user can contextualize her data by providing the creator's name, subject or topic of the data, project name, etc. Once that metadata is affiliated with any of the data objects, users can search on it.

7.1 What is Metadata?

"Metadata is often called data about data or information about information" [1, p. 1]. It describes the data in some way, such as providing information about the content, context of its origin or use, quality, condition, and associations to other data and objects in the world [2].

iRODS metadata is structured [1, 3]. When metadata is employed, an applicable scheme is also usually employed that defines descriptive elements and their ontological associations. For example, some data repositories use Dublin Core, a metadata scheme developed to describe web-based documents. Dublin Core elements include abstract, language, license, subject, creator, date, format, and publisher.

"Metadata can describe resources at any level of aggregation. It can describe a collection, a single resource, or a component part of a larger resource (for example, a photograph in an article)" [1]. Metadata can be embedded in a data object, or stored in a database and linked to the object it describes.

Metadata is used to facilitate data discovery—to improve search and retrieval. For example, suppose you upload a dataset to an online data repository for the purpose of sharing it with other professionals with similar research interests. If the system you upload it to doesn't support metadata, you can't include vital information about the creator of the dataset, the date when it was created, its purpose, its subject, any papers or findings associated with the data, any revisions made to the data, etc. Instead, users wishing to obtain your data will have to know the file name in order to search for it. If they don't know what you named it, they won't find your data. Someone browsing a variety of datasets and hoping to find one that covers the same subject as your dataset, or wishing to locate the data based on a paper they read that used the data, or hoping to find a dataset similar to theirs, but created more recently than theirs, will be out of luck. However, if you upload the data to a repository that supports metadata—as iRODS does—then users will be able to browse for or conduct a more targeted search for data like your own.

7.2 Types of Metadata

Metadata scholars often classify metadata into three categories: descriptive, structural, and administrative [1]. Descriptive metadata is intended to support data discovery and identification. Elements may include title, abstract, keywords, etc. Structural metadata describes the structure of the data object. For example, elements may allow metadata authors to describe components of the data object such as its title page, chapters, errata, how pages are ordered, number of pages, etc. Administrative metadata is intended to facilitate management and processing of the data. Elements could allow for identifying how the data was created, its file type, resolution, copyright information, licensing information, access privileges, etc.

Cornell University Library provides a nice chart describing these classifications of metadata: `http://preservationtutorial.library.cornell.edu/metadata/table5-1.html`.

Other classifications of metadata exist. Gilliland [4, p. 9] includes additional categories of preservation, use, and technical.

7.3 Why Metadata?

In addition to supporting data discovery, metadata also organizes and provides contextual and historical information about data objects, identifies structural relationships within and between data objects [4], identifies semantic relationships or differences between objects, "certifies the authenticity and degree of completeness of the content" [4, p. 6], distinguishes between different versions of data objects, provides legal support in the form of rights management, licensing, and reproduction information, enables users to assess authoritativeness and trustworthiness of the data through elements created to identify provenance, and facilitates interoperability, legacy resource integration, lineage tracking, and identification of persistent digital identifiers (e.g., Digital Object Identifiers, DOIs) to support preservation and archiving [1, p.1]

7.4 Metadata Generation and Storage

Metadata can be generated manually or automatically. In the case of manual generation, usually content contributors, librarians, or other information professionals create metadata for data objects. Manually generated metadata may be richer because manual methods exploit human understanding and judgment. For example, if a photographer is uploading her photos to a photo sharing website, she may be better at generating a rich description of the contents of the photo than a machine. She knows the story behind the photo, what objects within the photo are important to highlight, and how the photo fits into a larger context. A computer would have a harder time determining these things. However, manual generation is more time consuming than autogenerated metadata. It may be more costly if a librarian or other information professional is kept on staff to handle metadata. If busy data creators are expected to create metadata, it may be difficult to get them to adopt the practice and follow a standard scheme. Humans also make errors, such as typos and misspellings.

Automatically generated metadata is derived, extracted, or harvested.

Derived: The system knows certain things about any file it stores, such as creation date, file size, etc. This information is derived from the system and applied to the data object as metadata.

Extracted: In the case of extracted metadata, an indexing algorithm is used to pull information contained within the data object such as term frequency, subject/topic, noun phrases for author or title, etc. For example, if the data object is a journal article, the algorithm can employ natural language processing techniques to count the terms, identify co-located terms to suggest a subject or topic, or extract named entities such as the author's name.

Harvested: Metadata is aggregated from other sources, such as a metadata registry (e.g., Open Metadata Registry), database (e.g., The OAIster Database), or resource header (i.e., META tags) [3].

Automatic metadata generation also provides several benefits: speedy ingestion/extraction, costs incurred may be far less expensive (i.e., the cost of a few hours of a programmer's time), and no need to corral people into creating metadata and adhering to a standard. There are also drawbacks: computers don't possess human understanding or judgment beyond that written into the program, algorithms are designed to handle typical cases and may not be well suited to handle unusual cases, and computers make errors too (e.g., the determination of subject or topic is at best an estimation that one hopes reflects the truth).

7.5 Metadata Schemes

Metadata schemes are "sets of metadata elements designed for a specific purpose" [1, p. 2]. In addition to specifying metadata elements, schemes may also specify:

- rules for how content is formulated (e.g., how to identify a title),
- rules for content representation (e.g., capitalization rules),
- allowable content values (e.g., a controlled vocabulary used for the values of a subject or topic element),
- ontological and syntactical requirements for the associations or linkages between elements (e.g., whether an element is a class, property, data type, etc.), and
- encoding requirements (e.g., Standard Generalized Mark-up Language—SGML, Extensible Mark-up Language—XML) [1, p. 2].

References

[1] NISO, Understanding Metadata. Bethesda, MD: NISO Press, 2004.

[2] D. Hart and H. Phillips, "Metadata Primer — A 'How To' Guide on Metadata Implementation," National States Geographic Information Council, 1998. [Online].
Available: http://www.lic.wisc.edu/metadata/metaprim.htm.

[3] J. Greenberg, “Metadata and digital information,” in Encyclopedia of Library and Information Science, M. J. Bates, M. N. Maack, and M. Drake, Eds. New York, NY: Marcel Dekker, Inc., 2009.

[4] J. Gilliland, “Setting the stage,” in Introduction to Metadata, M. Baca, Ed. Los Angeles, CA: Getty Research Institute, 2008, pp. 1-19.

7.6 Using Metadata in iRODS

In iRODS, metadata can be used to describe data objects, collections, resources, and users. Metadata is stored as strings in the form of attribute-value-unit (AVU) triples, similar to those found in Resource Description Format (RDF). AVU triples are used for both derived metadata and user-defined metadata. For example, the photos in Alice’s collection already have some metadata associated with them—metadata that could be extracted from the data object and stored as an AVU triple. Let’s look at seal.jpg. The size of the file is 354 KB. Its dimensions are 1,664 X 1,664 pixels.

Attribute	Value	Unit
size	354	KB
dimensions	1,664 x 1,664	pixels

User-defined metadata about the contents of seal.jpg might look something like this:

Attribute	Value	Unit
animal	seal	
photo_color	grey and brown	
zoo	St. Louis Zoo	Missouri

In many metadata schemes, metadata attributes are defined by name-value pairs, similar to key-value pairs. As you can see from the user-defined example, an attribute-value pair is possible. Units are not required and can be empty strings. Units can also be used for some other descriptor, such as Missouri above.

7.7 The `imeta` Command

In iRODS, the main command line utility for handling metadata is `imeta`. It is used to determine, modify, list, search by, and delete iRODS metadata. Let’s use `imeta` with Alice’s account, because she needs to add metadata to her photos. So `iexit full` from the `rods` account, delete the rods environment file, and `iinit` for Alice. Do you remember how to do this on your own? Give it a try. If you get stuck, take a look at Logging In with Alice in Chapter 5: Using iCommands.

Let’s add an AVU to the collection `training_jpgs` that we created earlier with a recursive put. To create metadata for this collection we’ll use the add subcommand and the `-C` option, which signifies that we are adding metadata to a collection. (The `-d` option is used for adding metadata to a data object, `-R` for resources, and `-u` for users.)

```
$ imeta add -C training_jpgs collection_type jpg_photos
```

Order matters: Attributes are first, values second, and units third. So in the above example, the attribute we're adding is `collection_type` and the value we're adding is `jpg_photos`. We are not adding a unit.

Now, let's list the metadata using the `ls` subcommand for `training_jpgs` to see if `collection_type` and `jpg_photos` were added correctly.

```
$ imeta ls -C training_jpgs
AVUs defined for collection training_jpgs:
attribute: collection_type
value: jpg_photos
units:
```

Let's add the AVUs to individual data objects; but first, let's change our working collection to `training_jpgs`.

```
$ icd training_jpgs
```

Now, let's add the metadata. The first two AVUs will not have units, but the third will. Remember `-d` is for adding metadata to a data object.

```
$ imeta add -d seal.jpg subject wildlife
$ imeta add -d seal.jpg author AJones
$ imeta add -d seal.jpg size 354 kilobytes
```

To make sure these were added successfully, let's list the metadata for `seal.jpg`.

```
$ imeta ls -d seal.jpg
AVUs defined for dataObj seal.jpg:
attribute: subject
value: wildlife
units:
------
attribute: author
value: AJones
units:
------
attribute: size
value: 354
units: kilobytes
```

This confirms that the metadata has been added.

Recall that `imeta` also allows users to search on metadata. You may place conditions on attributes and values by using a comparison operator. For example, using the `qu` subcommand (meaning query), let's search for an object that has an author of AJones.

```
$ imeta qu -d author = 'AJones'
collection: /tempZone/home/alice/training_jpgs
dataObj: seal.jpg
```

The data object that has AJones listed as the author is `seal.jpg` in the `training_jpgs` collection.

Let's add metadata to another photo:

```
$ imeta add -d grapes.jpg subject food
$ imeta add -d grapes.jpg author AJones
$ imeta add -d grapes.jpg color_bw color
```

You may also search using wildcards. In iRODS, the percent symbol (%) is used as the wildcard character. Let's try this out. We'll search for data objects that contain any value for the attribute author, using the `qu` subcommand and the `%` wildcard.

```
$ imeta qu -d author like '%'

collection: /tempZone/home/alice/training_jpgs
dataObj: grapes.jpg
----
collection: /tempZone/home/alice/training_jpgs
dataObj: seal.jpg
```

From this search, two data objects were returned: `grapes.jpg` and `seal.jpg`. Both have an author attribute.

We can also search for collections based on their metadata. Let's search for collections that have an attribute of `collection_type`. Remember, for this we need to use the `-C` option.

```
$ imeta qu -C collection_type like '%'
collection: /tempZone/home/alice/training_jpgs
```

To modify a data object's AVU with `imeta`, we must specify the object's name, the attribute name, the attribute's associated value, and the desired new value. To change the author attribute from AJones to BSmith for `grapes.jpg`, we'll use the `mod` subcommand and the `v:` prefix for changing the value of author.

```
$ imeta mod -d grapes.jpg author AJones v:BSmith
```

To ensure AJones was changed to BSmith, let's list the metadata for `grapes.jpg`.

```
$ imeta ls -d grapes.jpg
AVUs defined for dataObj grapes.jpg:
attribute: subject
value: food
units:
----
attribute: color_bw
value: color
units:
----
attribute: author
value: BSmith
units:
```

Chapter 8

Workflow Automation

Without a tool like iRODS, processing large data sets must be done manually and can be tedious, complex, and time-consuming. With iRODS, you can save time and energy by creating powerful, customized workflows to process and perform computations on data objects. For example, when iRODS receives new data, the rule engine could be prompted to perform computations on the data, trigger actions within a High Performance Computing (HPC) system, or extract metadata from the data.

8.1 Rules

In iRODS, workflow automation is achieved through rules—scripts written in the iRODS rule language. The iRODS rule language contains its own native syntax, but provides a C-like structure which includes the basic capabilities of a procedural programming language: comments, native types, numeric and string operations, and function definitions. Advanced features such as regular expressions, list operations, dictionaries, and Language Integrated General Query (LIGQ) are supported. A hypothetical rule, `HelloWorld`, is shown below. This rule would print the text Hello World to the screen.

```
HelloWorld {
  writeLine("stdout", "Hello World");
}
```

Rules are executed by the iRODS rule engine—a built-in interpreter for the iRODS rule language. They may be triggered by

- the irule command,
- Policy Enforcement Points (PEPs), or
- invoking the `delay` directive in the case of Delayed Execution Rules.

The `irule` Command

A user may manually execute a rule that operates on data that she has access to by using the `irule` command; for example, to verify the checksums of data of a certain age.

Policy Enforcement Points

Policy Enforcement Points are triggers within the code that call specific rules. Editing these rules lets administrators change their data management policy. Later in this section, we will demonstrate the editing of `acPostProcForPut`, which is triggered after a newly uploaded data object is registered in the iCAT.

The `delay` Directive

The `delay` keyword is used to place rules into the delay execution queue rather than immediately executing the rule. Rules in the delay execution queue are monitored by the iRODS Delay Execution Server and are executed when desired.

8.2 Microservices

Rules may be extended by calling microservices. A microservice is the invocation of a C++ function which performs an advanced operation, such as accessing external libraries that are not available in the rule language like curl (used to access an HTTP resource), or performing operations that are computationally intensive such as image processing. Examples of microservices may include metadata extraction, image analysis, or the validation and verification of data at rest.

A hypothetical rule `HelloWorld2` that invokes a hypothetical microservice (`msihello_world`) is shown below:

```
HelloWorld2 {
  msihello_world(*msg);
}
```

8.3 Installing the Python Rule Engine

iRODS 4.2 added a rule engine plugin interface. Pluggable rule engines allow rules to be written in any supported language (including the original iRODS Rule Language). The Python rule engine will allow us to write rules in Python.

Multiple rule engines can be run concurrently and can call each others' rules. We will install the Python rule engine alongside the original iRODS Rule Language plugin and use them both in the next section.

To install the Python rule engine plugin (and a Python EXIF library we'll use):

```
$ sudo apt-get -y install irods-rule-engine-plugin-python python-exif
```

This rule engine package installs a single compiled library (shared object):
`/usr/lib/irods/plugins/rule_engines/libirods_rule_engine_plugin-python.so`

We will configure this new rule engine as part of the following metadata harvesting example.

8.4 The Example Rule: Harvesting and Applying Metadata with the Python Rule Engine

The example rulebases below (`training.re` and `core.py`) work together to harvest and apply metadata to any data object at rest, owned by any user. This rule could help Alice save time by automating metadata annotation for her. We will use this pair of rulebases to demonstrate the power of the rule engine plugin system. Using a text editor, copy the rulebases into new files in `/etc/irods` with the appropriate names.

```
# training.re
getSessionVar(*name, *output) {
    *output = eval("str($"++*name++")");
}
add_metadata_to_objpath(*str, *objpath, *objtype) {
    msiString2KeyValPair(*str, *kvp);
    msiAssociateKeyValuePairsToObj(*kvp, *objpath, *objtype);
}
```

```
# core.py
import os
import session_vars
import sys
import exifread
def acPostProcForPut(rule_args, callback, rei):
    sv = session_vars.get_map(rei)
    phypath = sv['data_object']['file_path']
    objpath = sv['data_object']['object_path']
    exiflist = []
    with open(phypath, 'rb') as f:
        tags = exifread.process_file(f, details=False)
        for (k, v) in tags.iteritems():
            if k not in ('JPEGThumbnail', 'TIFFThumbnail', 'Filename', 'EXIF MakerNote'):
                exifpair = '{0}={1}'.format(k, v)
                exiflist.append(exifpair)
    exifstring = '%'.join(exiflist)
    callback.add_metadata_to_objpath(exifstring, objpath, '-d')
    callback.writeLine('serverLog', 'PYTHON - acPostProcForPut() complete')
```

An implementation of the particular PEP we are going to look at already exists in the iRODS Rule Language rulebase that is shipped with iRODS (`core.re`); however we need the `acPostProcForPut` PEP to perform different behavior, so new logic has been written which will supersede the default behavior. This PEP is triggered after a data object is at rest (i.e., when all the bits have been uploaded and the data object is registered in iRODS), and this suits our purposes because after StockPhotoSite (SPS) data objects have been uploaded to iRODS, we will want the metadata to be harvested and applied to data objects for later discovery by SPS users.

There are two ways to insert the new PEP's behavior into iRODS.

1. The file `/etc/irods/core.re` could be edited with the new behavior (but this is inadvisable: `core.re` may change from release to release, so it might be overwritten during an upgrade. Also, we're using Python for this example and `core.re` is in the wrong rule language!), or
2. We can write the rule into an accessory rulebase (i.e., a file that contains a rule or rules but is not `core.re`) in `/etc/irods` and then tell iRODS to read this file by editing the

`/etc/irods/server_config.json` file. When we do this, the rules in our accessory rulebases (`training.re` and `core.py`), will combine to take precedence over the existing PEP in `core.re`.

Let's try the second approach. Open the file using a text editor such as nano:

```
$ sudo nano /etc/irods/server_config.json
```

Add a new python rule engine stanza within "rule_engines" just before the "irods_rule_engine_plugin-irods_rule_language-instance" stanza:

```
        {
            "instance_name": "irods_rule_engine_plugin-python-instance",
            "plugin_name": "irods_rule_engine_plugin-python",
            "plugin_specific_configuration": {}
        },
```

And add the new `training.re` rulebase to "re_rulebase_set":

```
                "re_rulebase_set": [
                    "training",
                    "core"
                ],
```

Save the file and the new rulebases will be activated! Let's upload a new file to our home collection:

```
$ wget https://github.com/irods/irods_training/raw/ugm2019/stickers.jpg
$ iput stickers.jpg
```

Let's look at the metadata that has been applied to `stickers.jpg`:

```
$ imeta ls -d stickers.jpg
AVUs defined for dataObj stickers.jpg:
attribute: Image Orientation
value: Horizontal (normal)
units:
----
attribute: EXIF ColorSpace
value: sRGB
units:
----
attribute: EXIF SubjectArea
value: [2015, 1511, 2217, 1330]
units:
----
attribute: GPS GPSDestBearingRef
value: T
units:
----
attribute: EXIF FNumber
value: 11/5
units:
----
<snip>
----
attribute: EXIF SceneType
value: Directly Photographed
units:
----
attribute: Thumbnail JPEGInterchangeFormatLength
value: 9289
units:
----
attribute: EXIF SubSecTimeOriginal
value: 738
units:
----
attribute: Image ExifOffset
value: 204
units:
----
attribute: Image YCbCrPositioning
value: Centered
units:
----
attribute: EXIF ExifImageWidth
value: 4032
units:
```

This simple example demonstrates automated metadata harvesting, use of multiple concurrent rule engines, and live changes to running server configuration. Now you're ready for the advanced training!

Appendix A

iRODS Resources

- irods.org — `https://irods.org/`
- iRODS overview — `https://irods.org/documentation/`
- iRODS download page — `https://irods.org/download/`
- iRODS GitHub site — `https://github.com/irods`
- iRODS documentation — `https://docs.irods.org/`
- iRODS LinkedIn Group — `https://www.linkedin.com/groups/8162245/`
- iRODS Twitter — `https://twitter.com/irods`

Appendix B

Glossary Of iRODS Terms

Agent An Agent is an instance of a server process that handles application programming interface (API) requests. Each time a client connects to an iRODS server, the server spawns an agent and a network connection is established between the agent and the requesting client.

Catalog Service Consumer (Resource Server) An iRODS server in a Zone that is not a Catalog Service Provider. There can be zero to many Catalog Service Consumers in a Zone.

Catalog Service Provider (iCAT Server, IES, or iCAT Enabled Server) The iRODS server in a Zone that holds the connection to the (possibly remote) iCAT. There is one Catalog Service Provider in a Zone.

Collection A Collection is the logical representation of physical containers, similar to directories or folders that are found in a file system. A Collection can have sub-collections, and hence provides a hierarchical structure.

Composable Resources Composable Resources are plugins that allow you to manage storage and retrieval of data on storage devices. There are two types of composable resources: Coordinating and Storage.

Control Plane The control plane receives status updates from all servers, and issues commands to servers to pause, resume, shut down, etc.

Coordinating Resource A Coordinating Resource is a type of Composable Resource that actively makes decisions about which physical storage device will receive or serve up a Data Object.

Data Object A Data Object is the logical representation of data that maps to one or more physical instances (Replicas) of the data at rest in Storage Resources.

Delayed Execution Rule A Delayed Execution Rule is a rule that invokes the delay keyword (i.e., a reserved word), which places the rule script in the delayed execution queue rather than immediately executing the rule.

Grid The hardware, operating system, and other machinery that support a Zone.

iCAT The iCAT, or iRODS Metadata Catalog, is a database (e.g. PostgreSQL, MySQL, Oracle) that stores metadata about the Data Objects, Collections, Users, and Groups in an iRODS Zone. There is one iCAT per iRODS Zone.

iCommands iCommands are Unix utilities that give users a command-line interface to operate on data in iRODS.

Microservice A microservice is a small, well-defined procedure that performs a server-side task and is either compiled into the iRODS server code or packaged independently as a shared object. Rules invoke Microservices to implement data management policies.

Policy Enforcement Point (PEP) A hook within the code of the iRODS Agent that invokes an interpreted rule script via the iRODS rule engine for the purpose of influencing a data management operation.

Replica An identical, physical copy of a Data Object.

Storage Resource A Storage Resource is the logical representation of—or pointer to—a physical storage device. They include the hostname and the directory path to the location of the Data Object on the storage device.

Vault The physical location of Data Objects on a storage device. For example, Vaults can be located on a Unix file system, a Ceph cluster, or on Amazon S3.

Workflow Some form of computation or action performed on Data Objects.

Zone An iRODS deployment, specifically the logical aspect of iRODS serviced by the iRODS Remote Procedure Call (RPC) application programming interface (API).

Zone Report A snapshot of an iRODS Zone, retrieved by using the izonereport iCommand.