OpenStack Trove Essentials

OpenStack has become an extremely popular solution to build public and private clouds with. A Database as a Service (DBaaS) enables the delivery of more agile database services at lower costs. Some other benefits of DBaaS are secure database deployments and compliance with standards and best practices. Trove is a DBaaS built on OpenStack and is becoming more popular by the day.

Since Trove is one of the most recent projects of OpenStack, DBAs and system administrators can find it difficult to set up and run a DBaaS using OpenStack Trove. This book helps DBAs make that step. We start by introducing the concepts of DBaaS and how it is implemented using OpenStack Trove. We will also look at implementing OpenStack and deploying Trove. You will learn how to create guest images and then look at how to provision databases in self-service mode, and how to perform administration tasks such as backup and recovery, and fine-tuning databases. At the end of the book, we will examine some advanced features of Trove, such as replication.

Who this book is written for

If you are a DBA, system administrator, architect, or a student who wants to build a DBaaS based on OpenStack, this book is for you. You should have a basic knowledge of OpenStack components, RDBMS/NoSQL, IaaS, and cloud computing.

What you will learn from this book

- Get to grips with the basics of OpenStack and the prerequisites to install Trove
- Understand the expectations of DBaaS and how Trove can help you achieve them
- Set up a basic installation of DevStack (Development Stack) in a virtual box
- Install Trove and utilize its configuration groups to manage and tune databases
- Use Image builder to create guest images for Trove
- Utilize Trove to provision your first database instance
- Back up and restore your databases with the help of Trove
In this package, you will find:

- The author's biography
- A preview chapter from the book, Chapter 7 'Database Backup and Restore'
- A synopsis of the book’s content
- More information on OpenStack Trove Essentials
About the Authors

Alok Shrivastwa is a technologist from India, currently working as the director of cloud services for Microland Ltd. in their Center of Innovation. He has a keen interest in all things physical and metaphysical and is an innovator at heart. He has worked with multiple large- and medium-sized enterprises, designing and implementing their network security solutions, automation solutions, databases, VoIP environments, datacenter designs, public and private clouds, and integrations.

He has also created several tools and intellectual properties in the field of operationalization of emerging technologies. He has authored a book, *Learning OpenStack*, with Packt Publishing, and has authored several whitepapers and blogs on technology and metaphysical topics, in addition to writing poems in Hindi. Also, he has been a guest speaker for undergraduate engineering students in Chennai.

You can connect with him at https://in.linkedin.com/in/alokas or follow him on Twitter at @alok_as.
Sunil Sarat is the vice president of cloud and mobility services at Microland Ltd., an India-based global hybrid IT infrastructure services provider.

He played a key role in setting up and running the emerging technologies practice, dealing with areas such as public/private cloud (AWS and Azure, VMware vCloud Suite, Microsoft, and OpenStack), hybrid IT (VMware vRealize automation/orchestration, Chef, and Puppet), enterprise mobility (Citrix Xenmobile and VMware Airwatch), VDI/app virtualization (VMware Horizon Suite, Citrix XenDesktop/XenApp, Microsoft RDS, and AppV), and associated transformation services.

He is a technologist and a business leader with expertise in creating new practices and service portfolios, building and managing high-performance teams, strategy definition, technological roadmaps, and 24/7 global remote infrastructure operations. He has varied experience in handling diverse functions such as innovation/technology, service delivery, transition, presales/solutions, and automation.

He has authored whitepapers, blogs, and articles on various technologies and service-related areas, is a speaker at cloud-related events, and reviews technical books. He has authored Learning OpenStack and reviewed Learning AirWatch and Mastering VMware Horizon 6, all by Packt Publishing.

He holds various industry certifications in the areas of compute, storage, and security and holds an MBA in marketing.

Besides technology and business, he is passionate about filmmaking and is a part-time filmmaker as well.

For more information, you can visit his LinkedIn profile at https://www.linkedin.com/in/sunilsarat or follow him on Twitter at @sunilsarat.
Database management has come a long way over the last decade or so. The process of provisioning databases used to start with racking and stacking a physical server, installing and configuring an operating system, and finally, installing and configuring a database management system. This entire process took weeks and, in some cases, months. Once the database is provisioned, you then of course have a whole host of things to be managed, including availability, backups, security, and performance. This provisioning and management consumed a lot of time and resources. During the evolution, we had two trends that have had a significant impact on the way databases were provisioned and managed. Automation eased the management aspect and virtualization eased the provisioning, at least up to the operating-system layer. Meanwhile, the other trend that we have seen is that enterprises are moving away from a single database technology model to a model which is fancily termed "polyglot persistence". This basically means adopting multiple database technologies with the intention of storing the data in a database that is best suited for that type of data. With multiple types of database technologies coming into play, enterprises are finding it difficult to manage this complexity while maintaining corporate standards and compliance.
Fortunately for us, over the last couple of years, cloud is the other trend that came to our rescue. With the advent of cloud, we have initially seen self-service based agile provisioning of infrastructure take off, which has been termed as Infrastructure as a Service and has automated a lot of aspects and made infrastructure management easier. Building on this a bit more, we now have self-service based agile provisioning of multiple types of databases, which is popularly known as Database as a Service (DBaaS). This has made things much easier for enterprises in terms of bringing in efficiencies and enforcing corporate standards and compliance. Enterprises can avail DBaaS from a public cloud such as Amazon Web Services or Microsoft Azure. Alternatively, they can build their own private cloud-based DBaaS and the need for this could be owing to various reasons such as data privacy and security. This is where OpenStack and Trove comes into the picture. OpenStack Trove is an open source implementation of DBaaS. While it has been in existence for a couple of years, it has started gaining momentum only recently with enterprises giving it a serious thought.

The benefits of DBaaS in general and OpenStack Trove in particular are obvious. The key challenge, however, is that beyond the documentation that is available from the OpenStack project itself, there is not much reading material out there to help potential DBAs and system/cloud administrators. This lack of skill and know-how is one of the potential inhibitors to OpenStack Trove adoption.

This book is an attempt to provide all the essential information that is necessary to kick-start your learning of OpenStack Trove and set up your own cloud-based DBaaS. In this book, the readers will be introduced to all major components of OpenStack Trove. Following this, the readers will get to understand how to set up Trove in both development and production environments, configuring it, and performing management activities such as backup and restore. Not to mention, it also deals with certain advanced database features, such as replication and clustering. This book takes a more practical approach to learning, as the learning from each chapter will contribute to the reader’s ability to build his/her own private cloud-based DBaaS by the time he/she completes reading this book. We hope you will enjoy reading this book and, more importantly, find it useful in your journey towards learning and implementing DBaaS using OpenStack Trove.

What this book covers

Chapter 1, *Introducing OpenStack Trove*, introduces the concept of Database as a Service and its advantages, followed by a quick introduction to the OpenStack Trove project and its components.
Chapter 2, *Setting up Trove with DevStack in a Box*, provides a list of prerequisites for the book. This chapter also helps you understand DevStack and its components and then helps you set up Trove with DevStack.

Chapter 3, *Installing Trove in an Existing OpenStack Environment*, gives you an overview of the different available methods to deploy Trove. It deals a little bit more in detail with installing Trove from source and the Ubuntu repository.

Chapter 4, *Preparing the Guest Images*, as the name implies, details how to build production-ready images that will be required by Trove.

Chapter 5, *Provisioning Database Instances*, looks at creating and launching instances using both CLI and GUI.

Chapter 6, *Configuring the Trove Instances*, introduces you to configuring Trove instances and also how to make configuration changes to multiple Trove instances using configuration groups.

Chapter 7, *Database Backup and Restore*, introduces the concept of Strategies and provides an overview of how to back up and restore Trove instances.

Chapter 8, *Advanced Database Features*, deals with advanced features such as replication and clustering in Trove.
Database Backup and Restore

Data being critical in every enterprise IT, it needs to be protected. This protection is done at various levels, by creating a cluster/replica to ensure more than one hot/warm copy of the data exists.

In order to have a cold copy of the data for disaster recovery, a database backup is normally taken. Database backups and restore are possibly one of the most important operational tasks of a DBA. Trove helps automate the entire process, from backing it up, encrypting data at rest, and also restoring the backup. Trove also supports incremental backups of your databases and supports creating a new instance from an existing backup.

In this chapter, we will cover the following topics:

- Formulating a backup and recovery plan
- The concept of strategies in Trove
- Configuration aspects
- Backing up and restoring Trove guest instances

Formulating a backup and recovery plan

There are two kinds of backups: full backup and incremental backup. Trove helps with both of the backups (with certain data stores), as we will see later in the chapter. A plan needs to be formulated in order to successfully execute these.
The frequency of both or either of these backups needs to be based on the following parameters:

- Importance of the data
- Frequency of changes in the database
- Recovery objectives (RTO, RPO)

Based on the preceding parameters, we will need to derive the following key points:

- Frequency of full backups
- Frequency of incremental backups (if any)
- Frequency of testing the backups (by restoring the database)
- Need for offsite shipping of the database backup
- Frequency and modes of offsite shipping

This plan is applicable to any form of backup and this will help us with scripting and automating the backup tasks.

**Backing up/restoring in Trove**

Trove uses backup strategies in order to back up the database. The backups are stored as defined in the storage strategy (defaults to Swift), which is at object storage system. The backup is encrypted by default to protect the data in rest.

There can be several use cases of backup and restore in Trove:

- Cold copy of data for recovery purposes
- Point in time snapshot in order to create a different branch of development
- Multiple copies of the database independent of each other for purposes like auditing, running reports based on old data, and so on

The Trove system internally uses backup and restore strategies to seed the replication data (discussed in the next chapter). Let’s now dive in and see how the backup/restore methodology works in Trove.

**The concept of strategies in Trove**

Strategy in the world of Trove means a construct that allows developers to extend the functionalities of Trove by writing specialized implementations that can be abstracted.
This is a fully pluggable architecture, and what this actually means is that different technologies and different codes can be used to perform the same functions across different database engines.

The concept of strategies is used for backups, restores, replication, clustering, and storage (this determines where the backups are stored along with its associated properties). These are implemented in the guest agent code (can also be implemented for the API and task manager components), which also makes the code run closest to the place where the action has to happen.

So, effectively, each strategy needs to implement a list of functions at a minimum (these can be seen in the `base.py` file for that particular strategy), which the system can then use to call and perform the functions.

For example, each backup strategy needs to provide a command that needs to be executed in order to take the backup, and each storage strategy needs to implement a `save` function, which will allow us to save to that particular storage system.

The following diagram shows the concept of strategies. It also shows that the control components use an abstracted term and send the message using the message bus, say `create_backup`, and the guest agent looks at the default or configured strategy for that particular database engine and executes those commands.
The concept is valid for everything that supports the strategies. Please note that not all the control components are shown in this case and the diagram is for representation purposes only.

The backup/restore strategy in action

In order to better understand how the strategy will work, let's take a look at the following diagram that shows the backup taking place. The steps are enumerated as follows:

1. The Trove API passes on the command to the Trove Task Manager.
2. The Trove Task Manager leaves a Message in the Rabbit MQ queue for the Guest Agent to pick up.
3. The Guest Agent pulls the message and checks the backup and storage strategy (configured/default) for the particular data store version.
4. The backup commands are executed by the guest agent. (It gets the command by the strategy definition.) For example, if the MySQLDump strategy is used, then the command executed is `mysqldump --all-databases --user <username> --password`, along with the command to zip and encrypt the backup (these are all defined in the strategy files (as shown in the next section)).
5. The Guest Agent stores the backup as stored in the storage strategy.
Configuring the backup strategies

The strategies are configured by default, but we can choose to override them. The configuration options are:

- **backup_strategy**: The name of the strategy to use, for example, InnoBackupEx, MySQLDump, MongoDump, and so on
- **backup_namespace**: The file to load the code for the strategies from
- **backup_incremental_strategy**: The name of the strategy that needs to be used while taking incremental backups

These configuration options are set in the `trove-guestagent.conf` file, which will inject them to the guest during build time.

We don't have to configure anything additional in the guest agent configuration; this section is purely informational.

In order to understand the different strategies available to us and the corresponding namespaces, let us take a look at the following table, which shows the different backup strategies that are available in Trove at the time of writing the book:

<table>
<thead>
<tr>
<th>Data store name / Backup type</th>
<th>Strategy name</th>
<th>Strategy namespace</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL / Full</td>
<td>MySQLDump</td>
<td><code>trove.guestagent.strategies.backup.mysql_impl</code></td>
</tr>
<tr>
<td>MySQL / Full</td>
<td>InnoBackupEX</td>
<td><code>trove.guestagent.strategies.backup.mysql_impl</code></td>
</tr>
<tr>
<td>MySQL / Incremental</td>
<td>InnoBackupExIncremental</td>
<td><code>trove.guestagent.strategies.backup.mysql_impl</code></td>
</tr>
<tr>
<td>Couchbase / Full</td>
<td>CbBackup</td>
<td><code>trove.guestagent.strategies.backup.experimental.couchbase_impl</code></td>
</tr>
<tr>
<td>Mongo DB / Full</td>
<td>MongoDump</td>
<td><code>trove.guestagent.strategies.backup.experimental.mongo_impl</code></td>
</tr>
<tr>
<td>PostgreSQL / Full</td>
<td>PgDump</td>
<td><code>trove.guestagent.strategies.backup.experimental.postgresql_impl</code></td>
</tr>
<tr>
<td>Redis / Full</td>
<td>RedisBackup</td>
<td><code>trove.guestagent.strategies.backup.expreimental.redis_impl</code></td>
</tr>
</tbody>
</table>
As we can see, at this point in time, only MySQL (and its variants like MariaDB) have the ability to perform the incremental backup and offer two strategies for full backup (if we choose not to use InnoDB, we could just use MySQLDump). Also, not all the different data stores support full backup at this moment.

This means that we can also implement a simple backup strategy of our choice, if we so choose, by writing a different Python class. However, in most cases, we don't have to as the ones provided by default with Trove are sufficient.

### Configuring the storage strategies

The storage strategy denotes the place where the backups can be stored. At the time of writing this book, only SwiftStorage, which is the object storage in OpenStack, has been implemented. The default configuration parameters are:

- **storage_strategy**: The name of the storage strategy
- **storage_namespace**: The file where this strategy is implemented

There are plans to add support for other storage strategies like AWS S3 and so on. But since this is the only strategy available to us at the moment, let us take a moment to also look at its sub-configuration parameters. The bucket, where the backups need to be stored, whether the backup needs to be encrypted, if it needs to be encrypted, what key needs to be used, and so on. All of these are configured using the following configuration variables:

- **backup_swift_container**: The place where the backups will be stored (default value is `database_backups`)
- **backup_use_gzip_compression**: Do we compress the backup (default is `true`)
- **backup_use_openssl_encryption**: Do we encrypt the backup (default is `true`)
- **backup_aes_cbc_key**: Which key to use for encryption
- **backup_use_snet**: Can the backup use the Swift service network (default is `false`)
- **backup_chunk_size**: Chunk size for backups
- **backup_segment_max_size**: Max size for each segment of the backup

Most times, the default would work fine. But these options can be configured should we need to tweak their values.
Backup prerequisites
The requisites for backup are fairly simple:

- We have a database for which backup has to be taken.
- The Swift system is configured and accessible. (In DevStack, please check if the Swift services are enabled.)
- There is space in the Swift system to store backups.
- All required configuration options are in place.

In our case, we don't have to worry about the last point as we will leave the entire configuration to the default values. Also, we have the second and third point taken care of. Since we are using MySQL as the database, the first point also has been satisfied. (Please note that while creating the DIB image, we installed the InnoDB tool by using the `ubuntu-mysql` element provided. So, we are good to go.)

Backups and restores
Now that we have established that the prerequisites are in order, we will perform the actual functions.

Full backup
The Trove command line with the `backup-create` option helps us create a full backup of the database.

Please note that backup/restore may turn off (or pause) the database service for a brief moment to ensure that data is not corrupted. So, caution should be exercised while taking backups or performing restores of production databases.

The command format is `trove backup-create <instance-id> <backup-name>`. Please remember that we can check the instance ID using the `trove list` command.
So, in order to back up the test12 database, we would execute:

`trove backup-create 723c048e-bd5b-4ela-84cd-836be970d7db myfirstbackup-1`

The details of the backup are shown on the screen. The backup command also backs up the metadata, which is especially useful while restoring or creating another database from the current backup.

**Incremental backup**

Incremental backup, as we know, is only supported for MySQL at this time, and we can perform an incremental backup of the database that we just backed up.

For the incremental backup to work, we will obviously need a parent or a full backup to anchor the incremental backup onto, so we will need the backup-id of a full backup to be used as a parent.

We will use the last backup we just took, whose id was `e3737982-2220-42a3-8e63-52d61f73f523` (it will be different for you).

The command format to take the incremental backup is similar to that of the full backup. The only exception is that a `--parent` parameter is being passed, so it will be `trove backup-create <instance-id> <backup name> --parent <parent backup-id>`.

So, we will execute the following command:

`trove backup-create 723c048e-bd5b-4ela-84cd-836be970d7db incremental-bkp --parent e3737982-2220-42a3-8e63-52d61f73f523`
Chapter 7

The output is similar, with the exception of parent_id being shown.

Viewing the backup

The backups can be listed by using the trove backup-list command:

```
trove backup-list
```

<table>
<thead>
<tr>
<th>ID</th>
<th>Instance ID</th>
<th>Name</th>
<th>Status</th>
<th>Parent ID</th>
<th>Updated</th>
</tr>
</thead>
<tbody>
<tr>
<td>e3737982-.....</td>
<td>723c48e-bd5b-.....</td>
<td>myfirstbackup</td>
<td>COMPLETED</td>
<td>None</td>
<td>2016-01-23T21:49:58</td>
</tr>
<tr>
<td>e8ba6800-.....</td>
<td>723c48e-bd5b-.....</td>
<td>incremental-bkp</td>
<td>COMPLETED</td>
<td>e3737982-2220-...</td>
<td>2016-01-23T22:00:56</td>
</tr>
</tbody>
</table>

The output shows that incremental-bkp has myfirstbackup as its parent. The backup is also stored in the Swift storage, so let us take a look at the Swift containers, by using the command swift list.

```
swift list
```

database_backups

As we can see, only the database_backups container is created. Please note that this is a default name for the Swift container and can be overridden by the configuration variables as shown in the previous sections.
Database Backup and Restore

If you get a user warning that states `UserWarning: Providing attr without filter_value to get_urls() is deprecated as of the 1.7.0 release` while executing the Swift command, you will need to set an additional environmental variable called `OS_REGION_NAME`.

Please set this to the default region name of your system. You can view this by executing the command `keystone endpoint-list` and then looking under the region. For us, it was `RegionOne`, so we export the variable with the following command:

```
export OS_REGION_NAME=RegionOne
```

We will then look into the container itself, by typing the command:

```
swift list database_backups
```

While we could technically download these backups elsewhere, please remember that these backups are encrypted (by default) and will be of no use outside Trove, unless the key is known.

**Restoring backups**

In Trove, the restoration of the database is not done directly, but by creating a new instance and loading the data onto it. We can restore from a full backup or an incremental backup. If we choose to restore from an incremental backup, the entire chain (up to the last parent full backup) is restored onto the system.

So, in order to create a new instance from a backup, we simply use the `trove create` command passing `--backup` parameter, `trove create <name> <flavor-id> --size <volume size> --backup <backup-id>`.

In our case, we will use incremental backup to ensure the full chain restore happens.

```
trove create copyoftest 2 --size 1 \
--backup e8ba6800-7ff0-40c8-9dd3-e396b84dd4f1 \
--datastore mysql --datastore_version 5.6
```
The new instance starts building and once it gets to the active state, we can verify that the same databases were found.

**Deleting backups**

The backup can be deleted by the trove backup-delete command by passing backup-id as the argument to the command. It is to be noted that we should only delete the full backups after all the incremental backups dependent on them are deleted. If the parent is deleted, then Trove automatically deletes the dependent backups as well.

So, here we delete the parent backup:

```bash
trove backup-delete e3737982-2220-42a3-8e63-52d61f73f523
```

When we execute a subsequent backup-list, we see that both the backups were deleted, as the incremental backup was dependent on the parent.
Summary
In this chapter, we dealt with backups and restores of the database. We have learned about the ways backup is implemented in Trove. In the next and final chapter, we will look at more advanced features such as replication and clustering.
Where to buy this book

You can buy OpenStack Trove Essentials from the Packt Publishing website.
Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

Click here for ordering and shipping details.