



Enterprise Database Migration

Migration Strategies

Rashmi Torgalmath
Customer Engineer,
Google Cloud



Hi, this is Rashmi again. Welcome back to Enterprise Database Migration.

There are a number of different strategies you can take when moving your applications and databases to the cloud. Let's talk about those now.

Learning objectives

- Lift and shift databases from on-premises to Google Cloud.
- Back up and restore databases from on-premises to Google Cloud services.
- Migrate databases to the cloud with no downtime.
- Optimize databases for the cloud.



- A lift and shift strategy involves picking up and moving virtual machines from one location into Google Cloud.
- Back up and restore is a traditional and simple way to move a database.
- To migrate databases to the cloud with no downtime, you can use database replication.
- Once moved, you should look for ways to optimize databases for the cloud.

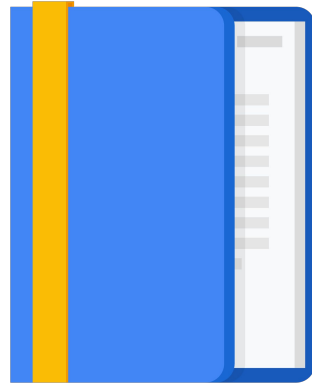
Agenda

Lift and Shift

Backup and Restore

Live Database Migration

Optimize Databases for the Cloud



Lift and shift means you are moving an application or database as is into your new cloud environment. This is often an easy and effective way of migrating databases and other applications as well.

Sometimes just moving a VM into Google Cloud is the right migration approach

Good candidates for lift and shift:

- Monolithic apps
 - A Wordpress site, for example
- Legacy apps
 - An ASP.NET app written years ago
 - Small, departmental databases
- Apps that are already virtualized on-premises or in another cloud

Steps:

1. Create an image of the virtual machine in the current environment.
2. Export the image from the current environment.
3. Copy the image to a Cloud Storage bucket.
4. Create a Compute Engine image from the exported image.



You may already be running your databases in virtual machines on-premises or in another cloud. Sometimes the best way to move it is simply importing the entire VM into the cloud and turning it on there.

- Monolithic apps, like a Wordpress site, for example, might be good candidates for a lift and shift approach.

Also, legacy apps like an ASP.NET app written years ago in an older version that can't be containerized and is not practical to rewrite.

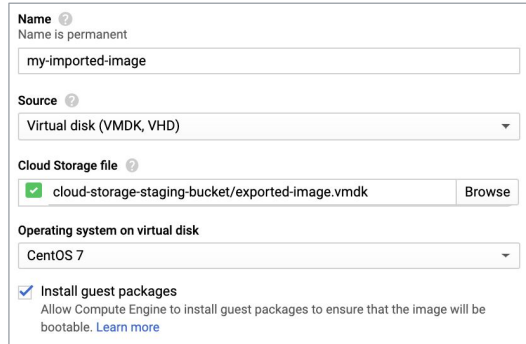
Small, departmental databases, apps that are already virtualized on-premises or in another cloud, and many other applications can be moved this way.

- In theory, lift and shift is simple. First, create an image of the virtual machine in the current environment. Then, export the image from the current environment and copy it to a Google Cloud Storage bucket. Last, create a Compute Engine image from the exported image.

Once you have a Compute Engine image, use it to create your virtual machine.

Google Cloud images can be created from external virtual disks

- VMWare .vmdk and Microsoft .vhd formats are supported.
- A Compute Engine VM is created to run the import and conversion job.



The screenshot shows a form for creating a Google Cloud image from a virtual disk. The form includes the following fields and options:

- Name:** A text input field containing "my-imported-image". Above the field is a question mark icon and the text "Name is permanent".
- Source:** A dropdown menu showing "Virtual disk (VMDK, VHD)".
- Cloud Storage file:** A text input field containing "cloud-storage-staging-bucket/exported-image.vmdk". To the right of the field is a "Browse" button. Above the field is a question mark icon.
- Operating system on virtual disk:** A dropdown menu showing "CentOS 7".
- Install guest packages:** A checked checkbox. Below it is the text "Allow Compute Engine to install guest packages to ensure that the image will be bootable. [Learn more](#)".



Google Cloud images can be created from external virtual disks. Both VMWare .vmdk and Microsoft .vhd file formats are supported.

This can be done from the console. A Compute Engine virtual machine is created automatically to perform the conversion.

Images can be exported from another cloud provider

AWS example:

```
aws ec2 export-image \  
  --image-id ami-0a0c02d1c4ac4d619 \  
  --disk-image-format VMDK \  
  --s3-export-location S3Bucket=s3-bucket-name-here,S3Prefix=exports/
```



If you're running a virtual machine in another cloud, first create an image of that VM. Then, you can export that image from the native Cloud Provider's format to a VMWare or Microsoft image.

In the example above, an AWS EC2 image is exported to VMWare format and saved in an S3 bucket. Once that is done, you can copy the image to Google Cloud Storage.

Caveats for migrating virtual machines

- Make sure you understand the license agreements for the software running on the VMs.
- Make sure you can log in to the VMs after they are moved.
 - For example, you might have to move SSH keys along with the image.
- Not always possible:
 - For example, images created from AWS Marketplace cannot be exported.

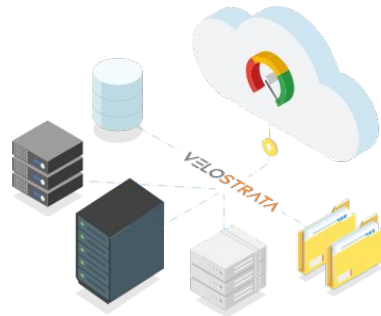


There are some caveats to this and it doesn't always work.

- Make sure you understand the license agreements for the software running on the VMs. You could unknowingly violate a license by moving it from one platform to another.
- Make sure you can log into the VMs after they are moved. You might have to move SSH keys along with the image so you can log into it when you run it in Google Cloud.
- Sometimes it's not possible to export a VM. For example, images created from AWS Marketplace cannot be exported.

To migrate many VMs, use Migrate for Compute Engine

- Field-proven
- VMware, AWS, and Azure sources
- Moves compute immediately (<10 minutes)
- Data is “streamed” to cloud until app is live in cloud.
- Testing and fallback/reversion are available.
- **For details, see VM Migration training course.**



Google Cloud Migrate for Compute Engine software will help you automate the migration of many VMs.

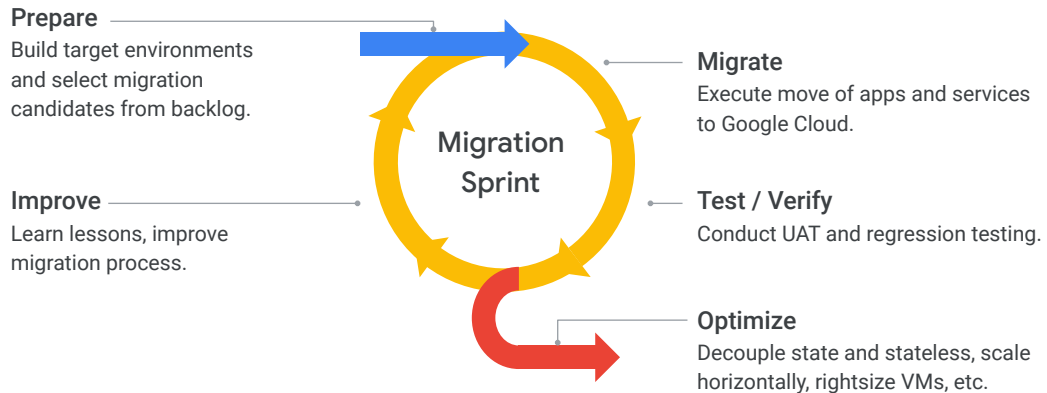
This works with VMWare, AWS, and Azure virtual machines.

Machines are moved very quickly and then their data is streamed into Google Cloud before it becomes live.

Because this is an automated system, you can have test runs and rolling back to the previous environment is easy.

This is a complex service, the details of which are beyond the scope of this course. There is a separate 3-day "VM Migration" training course available though.

Migrating VMs happens in sprints and waves



With Google Cloud Migrate for Compute Engine, VM migrations happen in waves.

Identify a set of VMs to migrate first. Prepare the VMs, Migrate them, Test to verify that they are migrated correctly. As you learn from your experience, look for ways to improve the process.

Once you've moved your VMs into Google Cloud, you can begin optimizing your applications and databases for the cloud.

Migrate for Compute Engine supports Windows and Linux virtual machines

Windows

Windows Server 2008 R2 SP1+
Windows Server 2012 and 2012 R2
Windows Server 2016 and 2019
Windows Server 2003, 2003 R2, 2008 R1 (offline)

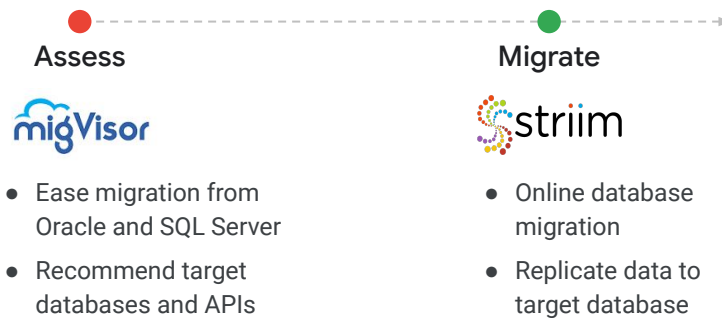
Linux

AWS Linux AMI, Linux 2
CentOS 6.4+ and 7
RHEL 6.4+ and 7
Debian 8.5+ and 9
Ubuntu 14/16/18; 12 (offline)
SUSE 11 SP3+, 12 SP2/3/4, and 15



Migrate for Compute Engine supports both Windows and Linux virtual machines. This slide lists the Windows versions and Linux distributions that are supported.

Invest in DB migration tooling with strategic partners



Large organizations are encouraged to invest in third-party migration tools and work with experience partners. Two tools that are popular are MigVisor and Striim.

- **MigVisor** is an assessment tool that helps identify dependencies and dependents and will recommend target services and databases based on the analysis.
- **Striim** is an online database migration tool. It supports many different database types as both target and source databases. It supports both homogenous migrations (Oracle to Oracle in Bare Metal Solution, for example) and heterogenous migrations (Oracle to Spanner, for example).

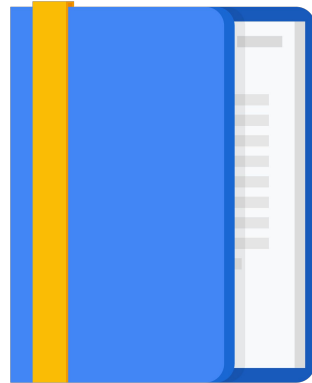
Agenda

Lift and Shift

Backup and Restore

Live Database Migration

Optimize Databases for the Cloud



There are different ways to migrate a database from an old server to a new one. The one you choose depends on different factors including the size of the database, how much downtime you can tolerate, and are you migrating to a new version of the database or even to a totally different database.

Back up and restore is the simplest way to move a database.

The simplest way to migrate a database is to back up from the source and restore it on the target

```
USE myDB;
GO
BACKUP DATABASE myDB
TO DISK = 'c:\tmp\myDB.bak'
WITH FORMAT,
    MEDIANAME =
'SQLServerBackups',
    NAME = 'Full Backup of myDB';
```

```
USE [master]
GO
RESTORE DATABASE myDB
FROM DISK = 'c:\tmp\myDB.bak'
WITH REPLACE;
GO
```



Perform a SQL backup on the source database, then copy the backup files into Google Cloud. Then, run a restore on the new target database server.

For large databases, use differential backups to minimize downtime

1. Start with a full backup and restore: this will take the longest.
2. Then, do differential backups until the backup-restore can be done quickly.
3. At some point, schedule a maintenance event when the database is unavailable.
4. Do the final backup and restore.
5. Migrate connections to the new database.
6. When all connections are migrate successfully, you can retire the old database.



For large databases, use differential backups to minimize downtime.

1. Start with a full backup and restore—this will take the longest.
2. Then, do differential backups and restores until it can be done quickly.
3. At some point, schedule a maintenance event when the database is unavailable.
4. Do the final backup and restore.
5. Migrate connections to the new database.
6. Assuming all goes well, you can retire the old database.

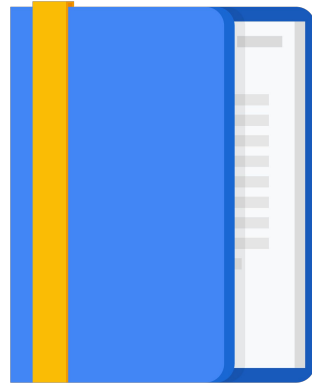
Agenda

Lift and Shift

Backup and Restore

Live Database Migration

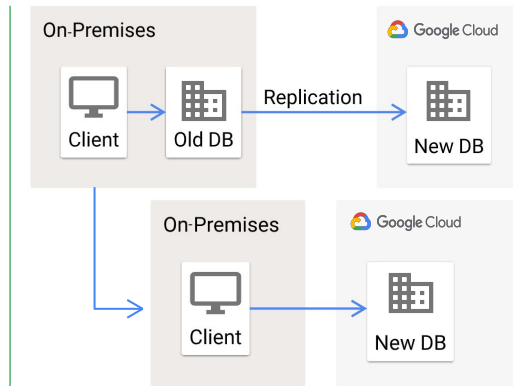
Optimize Databases for the Cloud



A live database migration means we switch databases with no downtime.

Database replication can minimize downtime compared to backup and restore

1. Configure the existing database as main.
2. Create the new database and configure it as the replica.
3. The main synchronizes the data with the replica.
4. At some point, migrate the clients to the replica and promote it to the main.



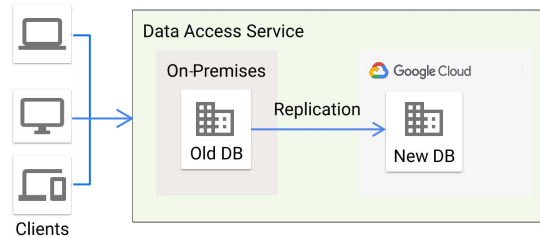
Using Database replication can minimize downtime compared to backup and restore.

- First, configure the existing database as main.
- Second, create the new database and configure it as the replica.
- Third, the main synchronizes the data with the replica.
- And fourth, after some time is passed migrate the clients to the replica and promote it to the main.

All modern databases support replication. This is mostly a database administration chore.

Using a data access service to migrate large numbers of clients can be seamless

1. Create a service that encapsulates all data access.
2. Migrate clients to use the service instead of connecting to the database.
3. When all clients are updated, the service is the only direct database client.
4. Replicate the database and then migrate the service connection.



Having a very large number of clients complicates the problem. Migrate large numbers of clients with no downtime by using a data access service.

- First, create a service that encapsulates all data access.
- Second, migrate clients to use the service, rather than connecting to the database.
- Third, once all clients are updated, the service is the only direct database client.
- Fourth, replicate the database and then migrate the service connection.

Use Blue/Green deployments to migrate data access services from on-premises to the cloud

1. Initially, a data access service exists on-premises.
2. Duplicate the service in the cloud.
3. Test the cloud service.
4. Use a reverse proxy or DNS to migrate client connections.
5. Can migrate back if migration fails.



Use Blue/Green deployments to migrate data access services from on-premises to the cloud:

1. Initially, a data access service exists on-premises.
2. Duplicate the service in the cloud.
3. Test the cloud service.
4. Use a reverse proxy or DNS to migrate client connections when ready.
5. You can switch connections back if migration fails.

Blue/Green deployments reduce the risk of a migration by allowing you to quickly revert back to the older service if a mistake is made.

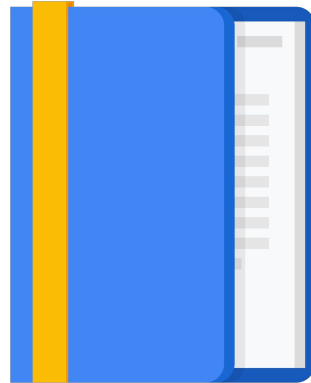
Agenda

Lift and Shift

Backup and Restore

Live Database Migration

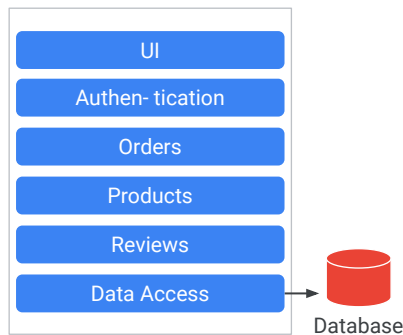
Optimize Databases for the Cloud



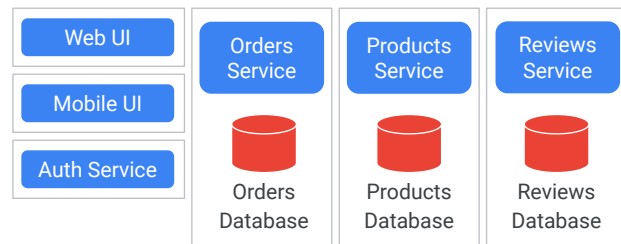
Most customers moving to Google Cloud are doing this so they can take advantage of its advanced features. Once you have your existing applications and databases migrated, you have an opportunity to optimize to take greater advantage of those features and the cost savings that the public cloud offers.

After the database is moved and its clients migrated, optimize for the cloud

Monolithic applications can be broken into smaller microservices.



Each microservice should have its own database.



Re-designing applications using a Microservice architecture is a good way to get started for optimizing your infrastructure. Large, monolithic applications are broken down into some number of smaller, independent services.

While many microservices might run on a single Kubernetes cluster, each should be programmed, deployed, and versioned separately. Also, each microservice should be responsible for its own data.

This requires large databases to be broken into smaller pieces, one for each service. To experienced DBAs, this might seem like a bad idea. It allows different services to use different types of databases that might be more appropriate and less expensive.

In the example pictured above, the monolithic application might start with an Oracle database for all data. However, when broken apart, maybe the Products and Reviews services could be migrated to a NoSQL database like Google Cloud Firestore. This would be significantly cheaper, easier to manage, and easier to program than Oracle. Maybe you would continue to use Oracle for the Order database, which would require a strong schema and ACID transactions.

Analyzing a monolithic database can help you determine microservice boundaries

Existing database data	Potential Google Cloud database service
Binary data stored in tables	Cloud Storage
Web application and session data	Firestore
OLAP data	BigQuery
Transactional data	Cloud SQL



When designing microservices, you are looking for ways to break up an application that minimizes the dependencies between the various microservices. Analyzing the monolithic database would help you determine where the microservice boundaries are. In a large database, you'll certainly find cases where groups of data are only loosely related, if at all. Finding these islands of data will help you determine how you can split off some functionality without negatively impacting the application as a whole.

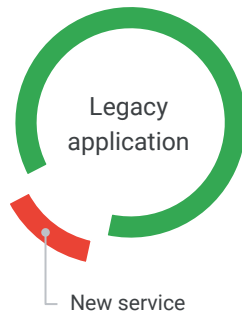
- If you are storing binary data inside your relational database, you might move that into Google Cloud Storage where it is significantly cheaper to store.
- You would often find data specific to web applications and web sessions inside a relational database. This type of data might be better suited for Cloud Firestore. Again, it would be cheaper, but also it would be easier to program for the web developers.
- Online analytical processing (OLAP) data can be moved from a relation system in Google BigQuery. BigQuery is massively scalable, easy to use and storage is plentiful and inexpensive.
- After the database is simplified, if you are using Oracle on Bare Metal Solution or SQL Server on Compute Engine, you might consider refactoring the application to use a managed database service like Cloud SQL or Spanner. This would likely make the total cost of ownership go down significantly by saving on administrative and licensing costs.

Use the strangler pattern to re-architect applications

Strangler pattern: Incrementally replace components of the old application with new services.

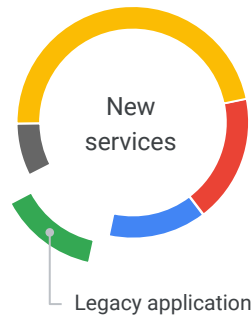
Early Phases of Migration

Strangler Facade



Later Phases of Migration

Strangler Facade



Don't try to refactor applications all at once. Rather, do it a piece at a time. Little by little, look for functionality that can be moved out of the larger application and into a separate service.

And where possible, use services already provided by Google. For example, if your monolithic application is responsible for authentication and authorization and your database is storing user information and passwords, refactor that to use a cloud service like Google Identity Platform. This will simplify your applications, be more secure, and likely save you money.

Martin Fowler named this idea of refactoring one component at a time the Strangler Pattern.

Module review



In this module, you learned the different migration strategies that you can use to move your applications and databases to the cloud. They are:

- Lift and shift when you want to simply migrate VMs directly to Google Cloud.
- Backup and restore from on-premises databases to Google Cloud.
- And live migration when you need to move databases to the cloud with zero downtime.

You also learned how you can optimize applications once they are moved to the cloud using the Strangler pattern.

