OpenStack Essentials:

DESIGNING, MIGRATING and DEPLOYING APPLICATIONS

A Guide to Cloud Applications on OpenStack

openstack.

Table of Contents

CONTRIBUTORS

Ricardo Ameixa, *Software Developer,* Volkswagen AG
Carol Barrett, *Cloud Software Planner,* Intel Corporation
Marcela Bonell, *Cloud Engineer,* Intel Corporation
Tyler Britten, *Technical Marketing Manager,* Red Hat
Kathy Cacciatore, *Consulting Marketing Manager,* OpenStack Foundation
Joanna H. Huang, *General Manager,* Aptira
Frank Kloeker, *Technology Manager Cloud Applications,* Deutsche Telekom
Amrith Kumar, *Founder, CTO,* Tesora
Mark Lamourine, *Sr. Software Developer,* Red Hat
Gerd Prüßmann, *Director Cloud Solutions,* Mirantis
Megan Rossetti, *Cloud Infrastructure Operations,* Walmart
Mark Smith, *Senior Product Marketing Manager,* SUSE
Yih Leong Sun, PhD, *Senior Software Cloud Architect,* Intel Corporation
Shamail Tahir, *Director of Product Management,* Athenahealth
Susan Wu, *Director of Technical Marketing,* Midokura

Introduction

Cloud-native. Containers. Container Orchestration Engines. Microservices. Bare metal. VMs. Patterns. OpenStack® software. How do they all fit together for software development teams?

We are all aware today's business climate is fast-paced and competitive, requiring mobility and massive reach. These characteristics drive organizations in all industries to software. New agile, cloud-native development technologies are transforming and creating new infrastructures and applications to explore new opportunities, making this an exciting time for application architects and developers.

New application frameworks such as containers require an open and programmable cloud infrastructure. OpenStack cloud software is the integration engine that enables your organization to take full advantage of the intersection of new application and infrastructure technologies. That's why OpenStack users can deploy new technologies quickly, and use already-integrated enterprise systems and networks.

> *Donnie Berkholz, Research Director, Development, DevOps and IT Ops, 451 Research, stated, "We're seeing container adoption within OpenStack users at … two to five times larger than container adoption outside OpenStack users, whether that's containers alone or container orchestration management tooling."*
>
> *Source: **"The Future of OpenStack + Kubernetes"** panel at CoreOS Tectonic Summit, December 2016; (https://www.youtube.com/watch?v=j4onxTI7m-k)*

OpenStack is governed by the **Four Opens** *(https://governance.openstack.org/tc/reference/opens.html)*: open source, design, development, and community. If you need a feature or service not already provided, you can collaborate and contribute it. OpenStack users welcome the opportunity to add capabilities that are rigorously tested with the entire platform—without the need to retrofit them internally with each new release.

Hundreds of the world's largest brands rely on OpenStack to help them move faster while lowering costs. OpenStack enables software-defined business environments for application developers and

architects with a single platform for application development and deployment with a stable and well-documented API, powerful yet simple SDKs, a global network of public clouds, and a thriving community.

This guide for application developers, architects and deployers was written by your business peers who develop for OpenStack clouds today. The guide includes recommendations on approaches, software development and migration patterns, OpenStack tools and services, and a planning checklist for moving applications from a public cloud to a private cloud. Discover why **the world runs on OpenStack** *(https://www.openstack.org/user-stories/)*.

Kickstart your journey—it begins with applications.

Comparisons of Traditional and Cloud-native Applications

Architecture comparisons

Historically, data center applications were designed for deployment on a specific physical server and operating system. This approach often led to server sprawl, underutilization of compute resources, operational complexity, and high capital costs. Deployment of traditional physical infrastructures was also time consuming and inflexible.

Virtualization of traditional applications initially addressed some of these issues. But now modern businesses require new levels of agility, flexibility and ease of management. The ability to pool and dynamically share cloud-based IT resources is necessary to allow developers to define and allocate infrastructure and resources based on workload needs. Using a cloud environment and designing cloud-centric workloads helps deliver new levels of agility, innovation and efficiency to maximize business value.

Characteristics of traditional applications

Traditional enterprise workloads or applications are often constructed in a three-tier architecture with a monolithic design. Most often, all of the application's functional components are bundled together. This form of application is the simplest and easiest to develop, test and deploy. However, when the application grows larger, the code becomes difficult to refactor because modules can be extensively dependent on each other. This approach often requires long-term commitments to a specific technology stack.

Scalability and high availability are often built into the underlying physical infrastructure, rather than being designed into the application itself. Hardware load balancers typically manage network traffic between and across the layers for performance and scalability. High availability and redundancy is usually provided by proprietary and expensive physical data center hardware. High performance, fast response times, and low latency are often delivered by close physical proximity or via expensive hardware solutions.

Characteristics of cloud applications

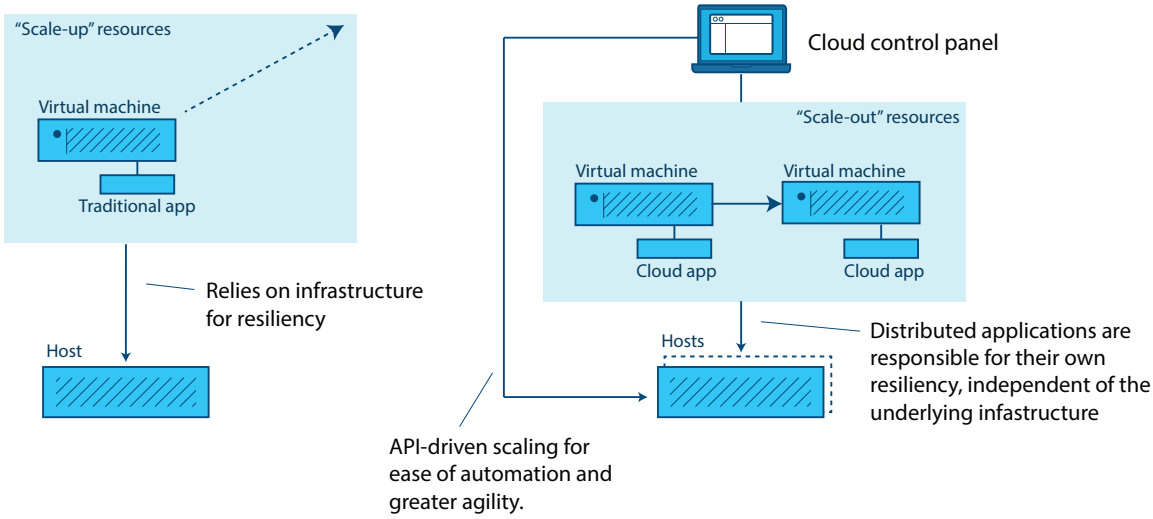
Cloud-native applications are designed to run in a software-defined infrastructure, where scalability, load balancing, high availability, and resiliency are delivered as part of the applications' architectural design. And advanced functionality is provided by software rather than expensive hardware alternatives.

These applications are resilient to failure, meaning the application can gracefully handle service interruptions caused by a physical infrastructure failure. The application can adapt to latency issues and is independent of the geographical locations.

Cloud-native applications are designed to scale on-demand. The fine-grained modularity and composable nature of cloud applications allows systems to scale specific features independently and isolate failures. These characteristics also enable developers to refine a part of the application without a complete re-write.

Figure 1 provides a high-level overview of the differences between application deployed in virtualized and cloud-based environments.

Figure 1: Virtualized vs. cloud application environments



The following table summarizes the characteristics of traditional and cloud-ready applications.

Traditional	Cloud-ready or cloud-native
• Monolithic and static • Tightly coupled • Specific and dedicated • Stateful • Powerful, complex, reliable hardware and software • Expensive, proprietary • Designed for never-break • Scale-up	• Highly modular • Loosely coupled • Stateless • Distributed architecture • Inexpensive, commodity hardware • APIs for services • Designed for failure • Designed for automatic scaling • Scale-out

Cloud benefits for migrated applications

BUSINESS BENEFITS

Organizations of all sizes must respond quickly to changing market and competitive landscapes. They must also improve efficiency, deliver innovation, and rebalance or reduce costs. Designing new cloud applications and migrating existing workloads to the cloud can help address all of these challenges.

However, it's important to note that delivering the benefits of a cloud-based strategy is not simply a matter of technology. Most business will also have to make significant changes to processes and cultures within their organizations.

TECHNICAL BENEFITS

OpenStack cloud provides a software-defined infrastructure for faster response times and greater agility. Developers and operations teams have self-service, on-demand access to compute, network and storage resources. This improves efficiency, removes IT bottlenecks, and delivers time to market advantages. Advanced functionality is also available via software instead of expensive proprietary hardware alternatives, delivering substantial cost reductions. Once an application is migrated to cloud-based architecture, it is possible to refactor a part of the system and allow each sub-component to scale independently.

Application Migration to the Cloud: Planning and Patterns

Application migration plan to the cloud

We recommend application architects and developers collaborate on a detailed, interactive migration plan for each application. The plan should be based on the selected best-fit migration patterns discussed in the table below after finalizing the services selected for migration. Communicate the plan and the decisions often with both technical and business stakeholders.

An application migration plan describes the context, objectives and challenges of the migration in addition to scenarios of how the applications will be used in the cloud. The main part of the plan should consist of coarse- and fine-grained migration paths (step-by-step, component-by-component) to the cloud. Migration paths ease planning and communications with stakeholders, project managers and cloud service providers. The plan should describe cloud migration patterns and the sequence to transform the overall system architecture and the application. Each step is identified by decomposing and rearranging multi-tier application services and combining them into groups of service components on the cloud. The integration of cloud services and migration objectives should also be considered. To help with reviews and validation, before and after architecture definitions and descriptions should be included in the migration plan.

Options for migrating applications to the cloud

An OpenStack cloud is ideal for developing and running new agile and innovative cloud-native workloads. Traditional applications can be transformed as well, offering cloud benefits to new and existing applications. Here are three application migration and deployment patterns to consider when preparing your plan.

CLOUD-HOSTED

Cloud hosting usually refers to deploying a traditional or conventional workload on the cloud while leaving the actual application and its architecture largely untouched. Applications designed for deployment on bare metal, physical servers, or virtualized environments, can be hosted on an OpenStack cloud without modification.

CLOUD-OPTIMIZED

In a cloud-optimized pattern, application elements can be modified to take advantage of cloud computing by utilizing software-defined capabilities to begin to reduce costs or improve efficiency, scalability, performance and functionality.

For example, a cloud storage system can be used instead of a traditional network file system. Or a cloud-ready database might be used in place of a traditional relational database. This approach can deliver improved functionality without redesigning the application.

CLOUD-NATIVE

Cloud-native applications are designed from the ground up to be deployed on a cloud-based environment. They can be extended as desired and are robustly built to protect against system failures and overload.

Traditional workloads can be transformed into cloud-native applications, if radically redesigned. This redesign work would need to be justified by measurable benefits such as availability and uptime improvements, improved scalability and flexibility, lower cost and TCO, or improved ROI.

Migrating existing applications to the cloud can be challenging and might require significant efforts, depending on:

- The overall application, system architecture, and structure.
- The dependency of the application on technologies used in the underlying infrastructure.
- Additional services integration required by the application's use case.
- Non-functional requirements or situational context of the application such as performance, availability, security, interoperability and legal regulations.

The best process for migration depends on the level of cloud-readiness or cloud-maturity of any given application. Migration pattern options include:

- A simple "lift & shift" approach.
- Partial deconstruction and restructuring of the application.
- Complete redevelopment of the application to a cloud-native approach.

The migration approach you choose will depend on the size and complexity of the workload or application. The process might include a sequence of steps, with gradually executed sections of modernization and deployment roll-outs. In some cases, the properties of the application need to be preserved, changed or enhanced.

This table compares potential application migration patterns to an OpenStack cloud in greater detail, including considerations dictated by the application's structure and architecture.

Patterns	Methodologies	Advantages	Notes
Cloud-hosted	Lift & shift, re-hosting. Move applications to cloud without changes in the compute model.	Make use of elastic cloud resources without changing application architecture.	Scalability only on application level. Benefiting from shorter re-deployment time. Application itself may remain a single point of failure.
Cloud-optimized	Cloudification, relocation, replacement. One or more components of the application are replaced with a cloud service rather than redeveloping the architecture of the application. For example: • Enrich the application with OpenStack cloud services such as the Trove DBaaS. • Replace specific storage services with elastic storage systems such as Swift object storage. In some cases, the application may stay on the former platform but uses services from the OpenStack cloud platform.	Optimizing applications for the cloud by using OpenStack cloud services: • Can increase availability, resilience, performance, speed of re-deployment. • Reduces capex. • Reduces time to market. Since a component is replaced by cloud services, no re-development efforts are needed.	New components and cloud services introduce APIs or protocols which were not used before. These might entail: • Modification costs. • Require training. • Added complexity. Most organizations feel the cloud and application modernization benefits more than outweigh the costs.

Patterns	Methodologies	Advantages	Notes
Cloud-native	Refactoring, modernization. Use the cloud to provide improved performance, scalability and elasticity to an application. A usage evaluation of components of a static or monolithic application is recommended. Complete rewrite of the application into cloud-native architecture.	• Optimal scalability, performance and availability. • Offers agile responsiveness to changing IT and business demands. Resiliency and availability are built into application. Provides horizontal scalability and elasticity.	

Application architecture considerations

This section discusses the evolution of mainstream application architecture design, including architectures and software design patterns to consider when planning your application cloud migration.

MONOLITHIC ARCHITECTURE

The term monolithic architecture refers to applications that tightly couple the presentation code, business logic, and data access tiers. This architecture was the predominant application pattern over the last two decades. Most monolithic applications are isolated and deployed on their host or virtual machine (VM). The benefit of this architecture is that most of the fault-tolerance is assumed to be in the infrastructure or systems layer. Application developers don't have to handle partitions and other availability constructs because it is assumed that all necessary data and services are available. The disadvantage is that the code base for the application is usually large and difficult to understand in its entirety.

SERVICE-ORIENTED ARCHITECTURE

Service-oriented applications were the precursor to and resemble microservices application patterns. In SOA, application services or components perform a specific function, such as executing business logic, or extract, transform and load (ETL) processes. They can communicate with one another over the network to invoke necessary application functionality. SOA services are accessible

to other application components through a communication protocol such as remote procedure call (RPC) or extensible markup language (XML) while still providing a single larger function. Each microservice is autonomous and independent but can still call one another when necessary.

MICROSERVICES ARCHITECTURE

Several software design patterns have emerged to help solve common problems. For example, the approach to decoupling a monolithic application into functional areas is well described by Chris Richardson's dissertation **Pattern: Microservices Architecture** *(http://microservices.io/patterns/microservices.html)*.

Applications developed with microservices involve components that are uniquely and independently implemented and upgraded. They can be written in a mixture of languages, using different backend databases, and yet, the system can be retired gracefully. Microservices enable developers to make their own technology choices such as integrated development environments (IDE) and software testing solutions.

STATEFUL VS. STATELESS

It is important to understand the differences between stateful and stateless applications when moving applications to cloud. In stateless applications, there is no state (session information, open files listing, etc.) for the server to manage. The client can retry requests because all the information needed is included or the function doesn't require inputs. For example, an HTTP request is a stateless exchange because all the information needed for the specific action is contained in the URL and sent with a POST.

Stateful applications generally rely on some information to be stored on the server side, which requires the client to communicate with the same server once a session has been established. An example of a stateful application is a Java® HttpSession, which provides a way to identify a user across multiple pages. The session is created between the client and server, requiring the client to communicate with the same server to take advantage of this capability.

In general, moving stateless services into the cloud is considerably easier than moving stateful services.

CAP THEOREM

As **mentioned by Professor Eric Brewer** in 2000 *(http://dl.acm.org/citation.cfm?id=564601)*, it is desirable to have Consistency, Availability and Partition-Tolerance when creating a distributed system. However, it is impossible to achieve all three. Based on the CAP theorem, and evidenced by many large-scale internet systems, it is impossible for a system to reliably provide consistent data when a network partition occurs in a distributed environment.

- ***Consistency:*** All nodes in a distributed system are guaranteed to return the most recent data for a given client request at all times.
- ***Availability:*** Every client request is guaranteed to receive a response in a distributed system.
- ***Partition-Tolerance:*** The system continues to operate when a network partition occurs (i.e. loss of network communication between the distributed system's processing nodes).

In large-scale distributed computing, such as cloud computing, network failures are not uncommon. Networks break down frequently and unexpectedly. Given that network connectivity is not 100 percent reliable, a cloud application system must be designed to tolerate a network partition condition (Partition-Tolerance). A cloud application design has to sacrifice one of the two remaining properties: Consistency or Availability. This results in either a CP (Consistency + Partition-Tolerance) or AP (Availability + Partition-Tolerance) cloud application.

CP (Consistency + Partition-Tolerance): A system is designed to choose consistency over availability. When the system enters partition mode, a portion of the affected nodes becomes unavailable and either returns errors or timeouts to client requests. The nodes resume activity only after the system has recovered from partitioning and the data is synchronized.

AP (Availability + Partition-Tolerance): A system is designed to choose availability over consistency. When the system encounters a partition event, all nodes continue to operate and respond to client requests by returning the most recent version of the data, even if it is inconsistent. The data will eventually become consistent after the system is recovered from partitioning.

When designing a cloud-native application, you have to choose one property over another in the event of a network partition (CP or AP). Building a cloud-native application involves multiple choices on architectural design. A CP design may be selected if an application mandates atomic operations. On the other hand, an AP design might be preferable if an application can be built around the concept of eventual consistency, and service availability is more critical to the business. It is important to note that the best design can be selected for different types of operations depending on your business needs and scaling requirements. Note, Eric Brewer's 2012 updates his original paper in, **"CAP 12 years Later: How the Rules have Changed"**. *(https://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed)*

In this chapter, we discussed the strategies of migrating applications onto an OpenStack cloud. Three common migration paths were described: cloud-hosted, cloud-optimized and cloud-native. Each migration option provides different advantages depending on the application needs and architecture. Guidance for a step-by-step migration plan was offered. We highlighted a few important architectural styles and factors for consideration while planning for the migration. Other technologies, such as containers (discussed in a later chapter), the **12-factor apps methodology** *(https://12factor.net/)*, CI/CD, and DevOps, are also useful information to be considered when migrating application into the cloud.

Cloud Infrastructure Considerations for Application Developers

Deploying to private, public or hybrid clouds

Several considerations must be addressed before determining the deployment model that best fits your application. Start by assessing your organization's regulatory compliance needs, financial resources, preferred spending model, users' geographic locations, and predictability of demand. Not all cloud models can equally address these requirements. These factors provide an initial guide to determining if a private, public or hybrid model best meets your application needs.

Private cloud may be the best option for applications with stringent regulatory requirements that dictate how data is handled and where it resides, or handling intellectual property or proprietary information. If the goal is cost-driven, maximum utilization of a private cloud may increase your return on investment. A private cloud also offers the ability to collaborate and discover new advancements that can be applied to your business needs and your teams' professional development. Examples of applications that are good fit for private clouds include financial applications, enterprise applications and big data.

A public cloud might be your best option during the initial application deployment when capex and physical infrastructure scalability are your main concerns. Many comprehensive resources are readily available from public cloud providers, including those running OpenStack, to help developers rapidly come up to speed. The total cost of consuming the cloud is more transparent because it is directly billed by the cloud provider. However, it is important to pay attention to the costs in addition to data and compute, including bandwidth, IP addresses, and disk I/O.

It is essential to be aware of the resource utilization (idling resources or right sizing) and cost forecasting for your applications. For example, you should know how much capacity your application uses as compared to how much is allocated to predict and control costs on a public cloud. Public clouds may also offer more unique or optimized capabilities such as GPU rendering or large storage. Enterprises often use public clouds to temporarily add resources to their private cloud during peak business times (also known as bursting applications), which is common for websites, eCommerce, and marketing campaign applications.

Many enterprises are realizing how a hybrid cloud can offer the best of both private and public cloud into a single strategy. Understanding how to use both private and public clouds—and the cost effectiveness of each—provides the most flexibility, scalability and agility. For example, hybrid cloud applications are ideal for stable applications that need to sporadically scale to handle extra workload

or reach users who are far away from your private cloud. As a developer, it's important to remember that applications might need to be portable and independent of private or internal systems. As an organization, hybrid cloud computing also provides the option to switch providers if the cost of private or public cloud becomes antithetical to the success of your business.

Changes in team responsibilities for modern applications

Historically, on-premises datacenter infrastructure and support teams delivered key application features such as: security, availability (uptime), capacity, and user response speed. Application teams deployed to this infrastructure to take advantages of these business-critical features. For cloud applications, these capabilities become the responsibility of the application teams.

Application teams are responsible for managing availability needed for a positive user experience. This includes providing the means for:

- ***Load balancing:*** Distributing the load across a cluster by starting additional instances of the application as needed. OpenStack LBaaS v2 (a Neutron service) provides an easy way for developers to accomplish this.
- ***Scalability:*** Increase the compute and storage resources dynamically. Together, OpenStack Orchestration (Heat) and the Ceilometer Telemetry services allow transparent auto-scaling based on CPU, memory, and disk usage.
- ***Failure handling:*** Detect application failure and restart the application without losing data and negatively affecting the user. For example, in case of an outage, the application should retry the user request on another running application instance instead of presenting an error message to the user.
- ***Disaster recovery:*** The standard APIs supported on all OpenStack Powered platforms allow easy migration to a secondary provider if needed.

In a traditional computing environment, the underlying infrastructure ensures security and trust. For cloud-native applications, the developer might need to implement methods to control who can access the application and data, what can be done with them, and how information is captured. The topic of authentication, authorization and auditing requires the attention of the application developers. It is also important to ensure data integrity over the full application lifecycle. Refer to the **OpenStack Security Guide** *(https://docs.openstack.org/security-guide/)* for additional information.

OpenStack APIs and SDKs for Application Developers

In a traditional environment, a system administrator provisions the infrastructure (e.g. bare metal servers, virtual machines) before it can run an application. If an application requires a feature that needs to store persistent files, a storage file system must be provisioned by the system administrator upfront. In the OpenStack cloud-based environment, the application programing interfaces (APIs) allow an application to perform advanced on-demand infrastructure provisioning as part of its logic. Depending on the use case, the application can dynamically configure the software to run computational tasks, based on user input parameters. This API-driven cloud-based model provides a mechanism for an application to dynamically control infrastructure in a way that is not possible with the traditional infrastructure model. The OpenStack community provides a comprehensive set of **developer resources and tools** *(https://developer.openstack.org/)*, including the details of the APIs and OpenStack SDKs, to help you write your first application, public and private development environments, reference architectures and more.

Application communication with OpenStack

INTERACTING WITH OPENSTACK SERVICES THROUGH OPENSTACK SDKS

Leveraging OpenStack services in your application is fundamental to managing and controlling cloud resources throughout the complete lifecycle of provisioning, managing, monitoring and reclamation.

OpenStack SDKs *(https://developer.openstack.org/)* are a set of software development kits for OpenStack that help cloud application developers write applications easily and quickly in their preferred programming language. By using OpenStack SDKs, developers can empower their applications to interact directly with cloud services through a language-level API, which follows the programming language standards selected for their applications.

AVAILABLE SDKS

SDKs are available in a variety of programming languages such as Python, Java, Ruby, Go, PHP and JavaScript. In terms of maturity, not all of them support the full set of OpenStack services, features, and versions. SDKs are not all maintained by the OpenStack community. They might be supported by an external community. Some SDKs, such as **Shade** *(https://docs.openstack.org/infra/shade/)*, work across all OpenStack clouds, while others, such as **fog** *(http://fog.io/)*, attempt to work across multiple cloud providers. For more information about available SDKs, visit the **OpenStack SDK wiki**. *(https://wiki.openstack.org/wiki/SDKs)*

The OpenStack User Committee is actively engaged in tracking and encouraging improvements to SDKs to enable application developers to write application across OpenStack clouds: You can follow the tracking and maturity of SDKs by becoming involved with the **User Committee's Working Groups** *(https://wiki.openstack.org/wiki/Governance/Foundation/UserCommittee)*.

How to choose an SDK

To select the best SDK for your application, determine what it needs from a cloud. Answer the following questions to find the SDK that best fits your application requirements:

- What are the programming languages used in your application?
- What services (compute, storage, etc.) does or will your application consume from the cloud?
- Will your application run in a hybrid cloud model such as a combination of OpenStack private and public clouds, AWS, Google Cloud Platform or Azure?
- After reviewing the maturity of the SDK and documentation to understand the completeness of features, does the SDK meet the application requirements?

Here are some of the most widely used SDKs:

Shade (Python): Shade is a simple client library for interacting with OpenStack clouds developed by the OpenStack Infrastructure team. Shade works across all modern OpenStack clouds and provides the most regularly-used services. *http://docs.openstack.org/infra/shade/*

Apache jclouds (Java): The Apache jclouds SDK is a multi-cloud toolkit that allows you to use portable abstractions or cloud-specific features for multiple clouds, including OpenStack, AWS, and Azure. *https://jclouds.apache.org/*

PHP OpenCloud (PHP): The PHP OpenCloud SDK enables PHP developers to easily connect to OpenStack APIs in a simple and idiomatic way. *https://github.com/php-opencloud/openstack*

Gophercloud (Go): Gophercloud is an open source library for working with OpenStack clouds in golang. *http://gophercloud.io/*

Fog (Ruby): Fog is a multi-cloud services library that provides a simplified interface, making clouds easier to work with and switch between. *http://fog.io/*

Adapting APIs as part of your development/application logic

Every OpenStack service presents an API such as Nova (compute), Cinder (block storage), Keystone (identity service), and Swift (object storage). Generally, OpenStack SDKs support a variety of OpenStack services and interact with the services' API. Using an SDK (or lower-level methodologies), application developers can integrate these APIs into their application design. This powerful approach is unavailable for traditional model applications.

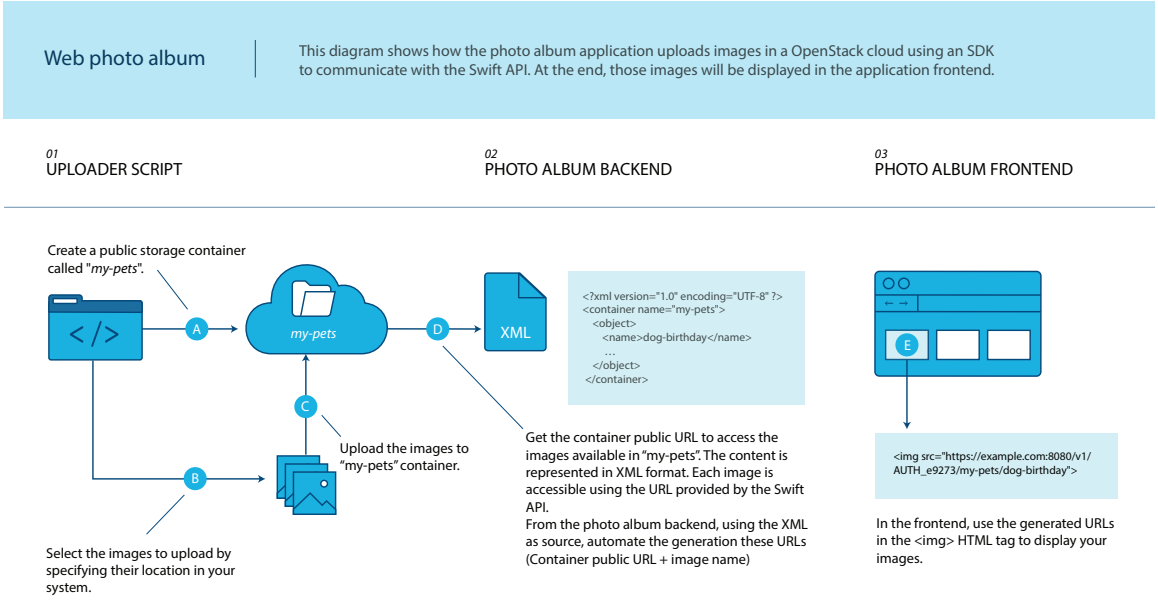
CLOUD STORAGE FOR APPLICATIONS

Swift, the OpenStack object storage service, can be used to serve large amounts of static data, such as image files, documents, and videos, through the standard HTTP protocol without web server involvement. As a developer, your application will interact with the service by using **object and storage containers APIs** *(https://docs.openstack.org/developer/swift/api/object_api_v1_overview.html)*. An object represents any static data. Storage containers are used to group those objects. Swift can also be used as a backup solution. In addition, other cloud technologies, such as the **Docker registry** *(https://docs.docker.com/registry/storage-drivers/swift/)*, support storage plugins for Swift object storage.

You can easily use the APIs to configure your application architecture based on ever-changing user expectations. In the next section, you will find useful examples for binding storage into your application through basic API calls to the storage services.

Figure 2 shows how a sample photo album application uses the Swift API to upload and store images in an OpenStack cloud. The web photo album displays those images from the cloud. The **code sample** *(https://github.com/MBonell/openstack-sdks-challenges/tree/master/shade/swift/photo-album)* uses **Shade** *(https://docs.openstack.org/infra/shade/)* (Python) as the SDK.

Figure 2: Example application using the Shade SDK to communicate with the Swift API

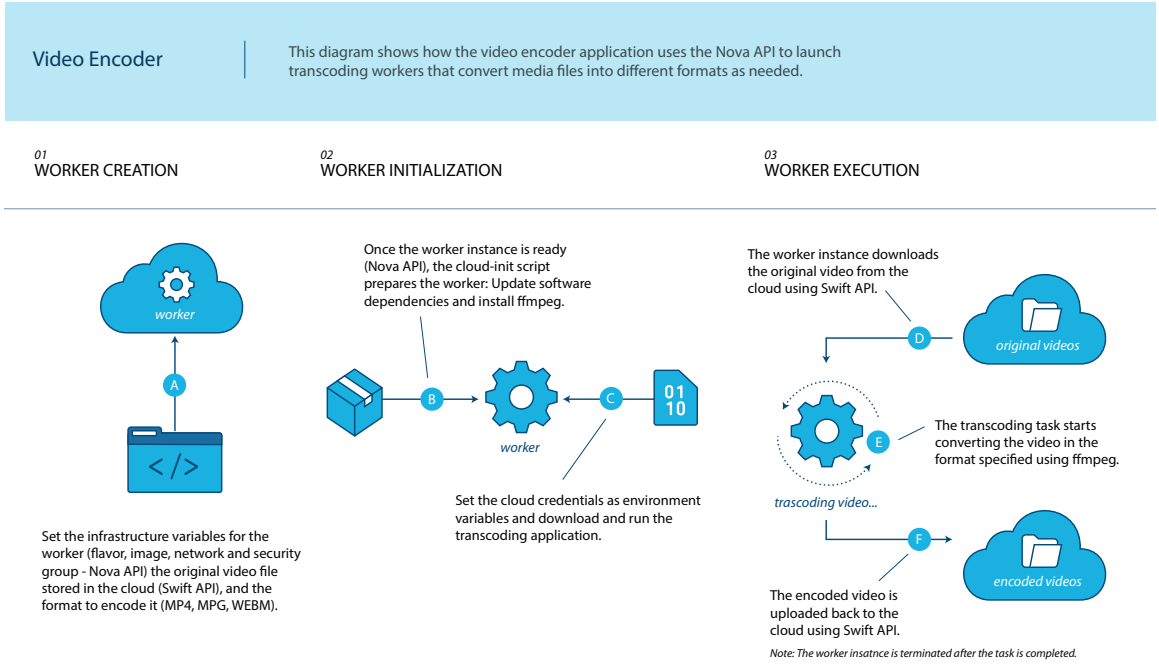


DYNAMIC COMPUTATION FOR APPLICATIONS

Some applications require dynamic execution of computational tasks such as batch processing or media transcoding. By using the SDK and interacting with the Nova API, developers can build applications that spin-up compute instances during runtime to perform computational or job processing tasks on-the-fly.

The following example shows how an encoder application uses the Nova API to launch transcoding workers to convert media into different formats. The **code sample** *(https://github.com/MBonell/openstack-sdks-challenges/tree/master/gophercloud/nova/encoder)* uses **Gophercloud** *(https://github.com/gophercloud/gophercloud)* (Go) as the SDK and **ffmpeg** *(https://ffmpeg.org/)* to transcode the video files.

Figure 3: Example application using the Gophercloud SDK to communicate with the Nova API



OpenStack Services to Support Application Development

OpenStack offers managed services that greatly improve the application development experience. They abstract the provisioning of dependent services, such as database-as-a-service, messaging-as-a-service or orchestration service. These services enable enterprise application developers to provide self-service access to the services or templatize a common development platform to streamline the development process.

Database-as-a-Service (DBaaS)

For application developers who want to leverage multiple database technologies, the complexities can be overwhelming. A properly tuned and updated database system is essential for applications that store crucial data. Provisioning and operating database systems with little tolerance for errors and misconfigurations is of paramount importance for application development.

OpenStack Database Service (Trove) provides a framework for successfully operating and provisioning a number of database technologies using best practices. Trove provides a consistent API for all commonly used operations throughout the database lifecycle from provisioning to configuration, tuning, operating, updating, backup, and more. Trove supports SQL databases such as PostgreSQL, MySQL, Percona, Percona XtraDB Cluster, MariaDB, DB2-Express, and Vertica, as well as NoSQL databases such as MongoDB, Cassandra, Couchbase, Redis and CouchDB.

Trove empowers application developers to provision these database technologies easily without knowing specific details of each. By providing close integration with OpenStack services such as Block Storage (Cinder) and Compute (Nova), operators can create resource pools and associate specific resources with development, QA and production systems. This advantage helps application developers to easily develop applications in a cost-effective environment, and be confident that when the database is moved to production, it will be operated in the same way. Trove also helps ensure a consistent database experience across a multi-region environment for application development.

Orchestration Service

OpenStack Orchestration (Heat) is a cloud orchestration engine that provides a template-based mechanism for launching a complete cloud application—from operating system through the user-facing presentation tier—on an OpenStack cloud. Heat offers features such as auto-scaling and nested application stacks.

Heat allows developers to describe the infrastructure and software components necessary for a cloud application in an easily readable text file that can be treated like code and version-controlled. Resources that can be described in Heat include OpenStack infrastructure components such as users, security key pairs, servers through Nova compute, disk attachment through Cinder volumes, firewall rules through security groups, or network configuration through Neutron networking. It also allows developers to define software components such as application configurations (scripts, Puppet manifests, Chef recipes) and software dependencies. It helps developers define composable and reusable software components. Software components can be defined once and deployed on multiple instances.

Heat supports implicit and explicit dependencies. An implicit dependency is automatically enforced when a property or input of a software deployment is obtained from the attribute or output of another deployment. Timing dependencies can also be specified. For example, a database component must be up and running before an application component can connect to it. Developers can simplify debugging and troubleshooting by creating and sharing common Heat templates with other developers to provision consistent applications.

Although it is possible to define all infrastructure and software definitions into a single template file, it is recommended to split the definitions for large and complex scenarios into multiple Heat files. Heat offers nested stacks to support this solution, making the templates reusable and easier to read and consume.

Heat provides an auto-scaling feature that should be integrated with the OpenStack Telemetry engine (Ceilometer). The application can automatically scale up and down. When a resource is defined in a scaling group, Heat automatically creates and configures the required software components in the correct order based on the scaling definition specified by the developers. It automatically re-balances the resources when the load is decreased.

Heat also supports application lifecycle management. You can update and change your infrastructure and software deployment by updating an existing stack with a modified template. Heat will automatically make the necessary changes to the system based on your new definitions.

Heat allows developers and operators to collaborate in an agile DevOps environment. It enables the team to define infrastructure as code and integrate it as part of the application development lifecycle. This approach improves infrastructure proximity across various software development lifecycle stages (Development, QA, Production), reduces application deployment issues and concerns, and increases software quality.

Application Catalog Service

Murano is the OpenStack application catalog service that enables users to browse and find applications available to deploy on the cloud. It is designed to provide an integrated and automated

turnkey deployment solution for cloud applications. Murano also allows application developers to compose reliable application environments and maintain the application service consistency across cloud projects.

Murano simplifies cloud application deployment, and manages application lifecycles to support different service workloads ranging from common use cases to complex, large-scale, and distributed applications. The catalog service is fully integrated with OpenStack services including Identity, Orchestration and Dashboard so users can generate complete applications with all the compute, storage, networking, and application resources in place.

One use case example involves launching a LAMP (Linux, Apache, MySQL, PHP) stack on the cloud. The required application packages for the LAMP service bundle are developed and stored in Murano. Cloud users can take advantage of those pre-defined packages to automatically deploy web services with a couple of clicks, using the OpenStack Dashboard or Murano command line interface (CLI).

Murano also assists in DevOps collaboration between developers and operators by generating visibility and transparency across various deployment cycles.

Messaging-as-a-Service

Zaqar is an OpenStack project that provides a multi-tenant cloud messaging and notification service. Developers use Zaqar within applications to communicate with end users or other software agents. Typical use cases include event broadcasting, task distribution, point-to-point messaging or email notification. Zaqar provides two communication protocols: HTTP-based RESTful API and WebSocket-based API. The HTTP-based API is firewall-friendly and provides simple request/response-style communication. The Websocket API provides communication over a persistent connection and transfers multiple request/response communications over a single TCP connection, which potentially reduces network traffic and minimizes delays.

As OpenStack continues to evolve and expand, more innovative services are developed and added such as Magnum (Container orchestration-as-a-service), Manila (File system-as-a-service), Barbican (Key management-as-a-service), and more. The **OpenStack Project Navigator** *(https://www.openstack.org/software/project-navigator/)* provides useful insights such as maturity characteristics on OpenStack projects.

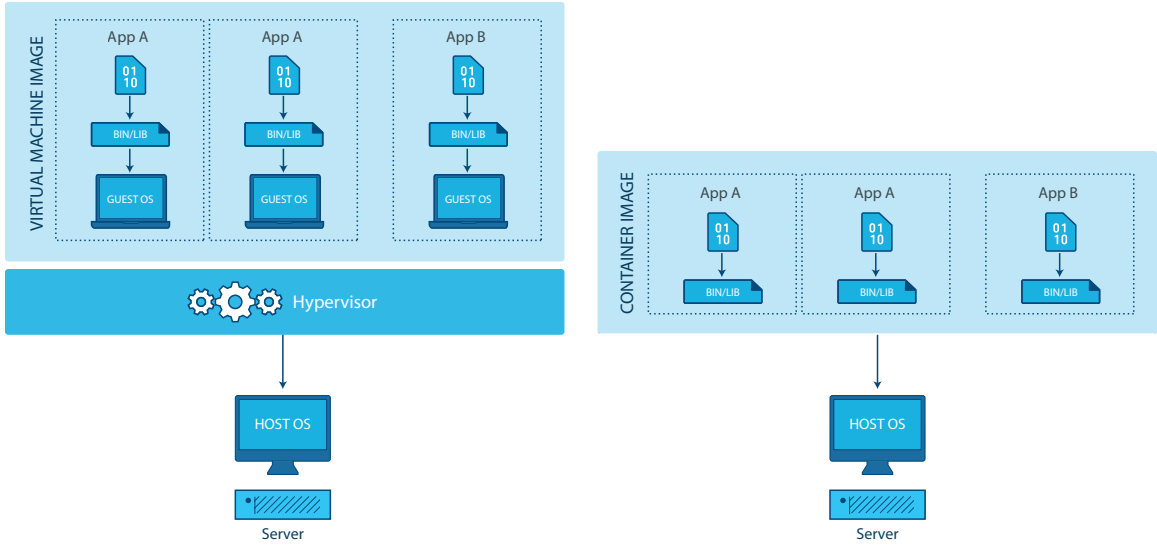
Containerizing Applications on OpenStack

Virtual machines and containers

In a traditional virtualized environment, a VM includes a full operating system and often runs on a specific hypervisor such as KVM, XEN, HyperV or ESXi. To write and deploy applications for a virtualized environment, the entire application is packaged, including the binaries, libraries, and operating system, during the deployment process. Virtualization makes efficient use of hardware resources and provides a high degree of isolation. However, a VM is heavily weighted, which makes it challenging to maintain environment parity across development, testing and production.

A container runs in an isolated process on a host operating system. It does not include full operating system libraries, but it does provide a lightweight environment by sharing kernel resources with other containers. Containers are portable and run exactly the same across development, testing and production environments. Figure 4 shows the fundamental difference between VMs and containers in any environment. An introduction to containers in OpenStack is provided in a **white paper** *(https://www.openstack.org/assets/pdf-downloads/Containers-and-OpenStack.pdf)*

Figure 4: Virtual machine and container architectures



Containerizing applications

It is not always necessary to completely redesign a whole system to fit into containers. An application can be partially containerized. The portion of the system that is long-lived or depends on large-scale private storage, such as a database, can remain on bare metal or in a VM. If developers adopt a microservices architecture design, containers are one of the best approaches to package and deploy the individual services.

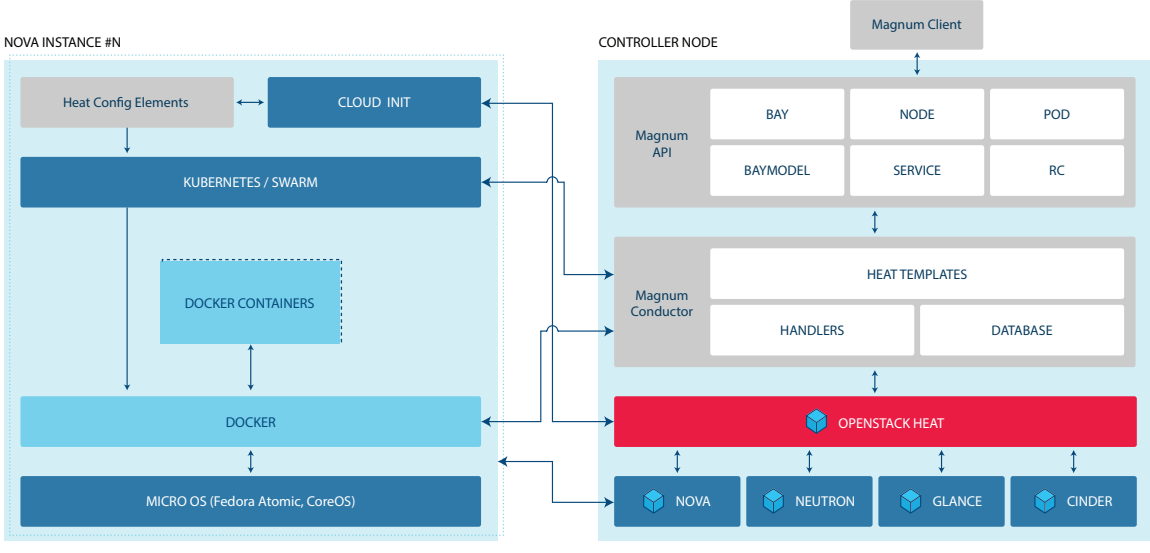
A number of container technologies are available today including Docker, LXC, LXD or rkt. Each technology provides a mechanism to package and install an application in its containers. For example, a Dockerfile specifies a Docker container along with commands to install the application, software or other environment variables.

Generally, a complete system will consist of multiple containers for deployment on a cluster of machines. Container orchestration tools such as Docker Swarm, Kubernetes, and Apache Mesos are used for the deployment. While each of these technologies have their differences in architecture and terminology, they follow a common pattern—all container deployments are described in the manifest file. The orchestration tools manage the container placement, scaling, networking, discovery, updates, and other services.

Deploying and operating containers on OpenStack

There are a number of options for deploying a container cluster on an OpenStack cloud. One option is the Container Orchestration service (Magnum). It allows a user to request a Mesos, Docker Swarm, or Kubernetes cluster from the OpenStack Dashboard (Horizon user interface), CLI, or API. Magnum allows multiple container technologies to be used concurrently in OpenStack.

Figure 5: OpenStack Magnum container architecture overview



Containers started by Magnum are run on top of an OpenStack resource called a bay. Bays are collections of Nova instances created with Heat. Magnum uses Heat to orchestrate an OS image that contains Docker Swarm, Kubernetes or Mesos, and runs that image in either virtual machines or bare metal in a cluster configuration. Magnum simplifies the integration with OpenStack, and allows cloud users who can already launch cloud resources such as Nova instances, OpenStack Block Storage (Cinder) volumes or OpenStack Database Service (Trove) databases to create bays where they can start application containers.

Magnum is not the only option for deploying a container cluster on OpenStack. A number of third-party tools, such as Ansible and Terraform, can be used to deploy container clusters and automatically deploy resources using the OpenStack API. It's up to individual organizations to choose the approach that makes the most sense for their environment.

Moving Applications from Cloud to Cloud

Cloud computing provides an array of hosting and service options to fit your overall company strategy. Sometimes a public cloud is your best option and other times your data requirements demand a private cloud. As needs converge, a hybrid solution continues to gain popularity. Developers must consider if their applications might be run on either or both. This chapter discusses considerations when moving applications from cloud to cloud and additional models such as cloud bursting. It assumes your organization has researched financial implications, and has made the determination to move applications between clouds. Good sources for financial analysis include, but are not limited to, **OpenStack: A Business Perspective** *(https://www.openstack.org/assets/pdf-downloads/business-perspectives.pdf)* and 451 Research April 2016 presentation, **"OpenStack Pulse: Unbiased Research on Enterprise Demand, TCO, and Market Size"**. *(https://www.openstack.org/videos/video/openstack-pulse-unbiased-research-on-enterprise-demand-tco-and-market-size)*.

Decoupling applications from the public cloud

Decoupling basic web applications that are deployed on a public cloud can be straightforward. A domain name server (DNS) redirect and a simple file copy is generally sufficient to move a web application to a private cloud.

The situation becomes more complex if the application is leveraging the public cloud's managed services such as DNS, elastic load balancing (ELB), databases, monitoring, security or if the application deployment is tightly coupled to the public cloud environment. Application developers should have a migration plan for transferring the data in a way that avoids application downtime. Another consideration is financial—the exit cost—where a public cloud provider charges for the data transfer out of the public cloud. For example, application owners should be aware of charges for data transfers between regions within the same public cloud.

Networking and latency concerns

When moving applications from a public cloud to a private cloud, consider the infrastructure requirements for networking, security and data. One approach is to mirror the networking architecture from the enterprise data center in the public cloud and the on-premises private cloud. A common use case is to implement custom networking configurations such as L2 adjacency. Also consider leveraging distributed computing concepts such as share-nothing architecture to ensure

partition-tolerance and application availability. If you are accustomed to having dedicated links from the public cloud to the corporate environment, be sure to review bandwidth requirements, especially if there is a significant volume of data shared across the organization and remote offices.

Mapping public cloud services to OpenStack services

Public cloud providers offer managed services that application developers might leverage in their applications. When moving applications from a public to an OpenStack private cloud, a common scenario, be sure to understand the dependencies and map them to equivalent OpenStack services. The following table maps the most commonly used services for OpenStack and popular public clouds.

Service	OpenStack Project	Amazon Web Services	Microsoft Azure	Google Cloud Platform
Virtual servers	Nova	EC2	Virtual Machines	Compute Engine
Block storage	Cinder	Elastic Block Store (EBS)	Disk Storage/ Page Blobs	Persistent Disk
Object storage	Swift	S3	Blob Storage	Cloud Storage
Orchestration	Heat	CloudFormation	Resource Manager	Deployment Manager
Database	Trove	RDS	SQL Database	Cloud SQL
Messaging	Zaqar	Simple Queue Service (SQS)	Service Bus Queues	Cloud Pub/Sub
Containers	Magnum	EC2 Container Service	Container Service	Container Engine

For more information, visit

- **OpenStack Project Navigator** *(https://www.openstack.org/software/project-navigator/)*
- **Amazon Web Services** *(https://aws.amazon.com/servicecatalog/)* (free account required)
- **Microsoft Azure** *(https://docs.microsoft.com/en-us/azure/guidance/guidance-azure-for-aws-professionals-service-map)*
- **Google Cloud Platform** *(https://cloud.google.com/free-trial/docs/map-aws-google-cloud-platform)*

Data and data residence policy decisions

Whether your company decides to use a public, private or hybrid cloud model, consider the legal implications regarding data collection, storage or processing. There are likely state, national, country, and international laws and regulations that need to be considered to ensure legal compliance. Industry-specific regulations may affect your plan also; two examples are the **Payment Card Industry Data Security Standard** *(https://www.pcisecuritystandards.org)* (PCI DSS, global cardholder safety), **Health Insurance Portability and Accountability Act** *(http://www.hhs.gov/hipaa/index.html)* (HIPAA, a United States law). Consult your legal counsel to ensure your business is and remains in compliance.

Cloud bursting into public cloud from private cloud

Cloud bursting is an application deployment model in which an application workload running in a private cloud bursts into a different cloud environment to meet dynamic workload demands. Cloud bursting is typically used to handle a traffic spike tied to a seasonal demand, a news event, or special offer that drives traffic to an application for a specific time period.

Under this hybrid cloud deployment, an organization only pays for the extra compute resources when they are needed. For example, an organization might need to maintain the application state in the private cloud while bursting the traffic spike to public cloud on-demand resources. A hybrid model with bursting also provides an excellent environment for application performance and stress testing. Latency between the two environments is an important part of successful cloud-bursting because of its ability to impact performance for the entire application system.

Checklist: Public to Private Cloud Migration

Before making the decision to migrate your application environment to the cloud, consider reviewing this checklist.

1. Workload discovery in the public cloud:
   - **a.** Inventory all the applications in the public cloud.
   - **b.** Inventory the sizing of each application in the public cloud (cores, memory, storage on each instance) and map it to the right instance type in the private cloud.
   - **c.** Estimate the cost of running the applications in public cloud (include IaaS, data transfer cost, software licenses, and support cost).
   - **d.** List the regions where the applications are deployed (North America, Europe, Asia, etc.)
   - **e.** List of the disaster recovery requirements (e.g., RTO – Recovery Time Objective and RPO – Recovery Point Objective).
   - **f.** Inventory all VPC network and security requirements.
   - **g.** Inventory all third-party software in use.
   - **h.** Inventory all the management software used to manage workloads in the public cloud (Active Directory, DNS, monitoring, deployment, backups, disaster recovery replications, ticketing and alerting systems, etc.).
   - **i.** Review the application and data dependency matrix—the application's dependency on any other applications for connectivity, security, integration and data size.
   - **j.** Understand or benchmark the performance of the application/workload in the public cloud.
   - **k.** Understand the public cloud native service dependencies:
     - **i.** DNS service.
     - **ii.** Storage dependencies (object storage, archival, block storage, etc.).
     - **iii.** List any deployment and automation services.
     - **iv.** Map all the database services being used (include relational or NoSQL, data warehouse).
     - **v.** Understand if any notification, queuing, or email services are in use.
2. Planning the workloads for the private cloud:
   - **a.** Based on the sizing exercise in the public cloud discovery, understand the capacity needs for the workloads running in the public cloud (e.g., CPU cores, memory, storage and networking).
   - **b.** Based on the workload characteristics on the public cloud, consider cloud-native architectures such as microservices or containers.

   c. Based on the regions in which the applications/workloads are currently deployed, evaluate the data center cost and colocation options.
   d. Based on the native public services in use, map alternative software and technologies to remove the dependencies from the public cloud.
   e. Plan proof-of-concept (POC) and performance benchmarking exercises for the selected applications/workloads in the private cloud infrastructure.
   f. Complete a costing exercise including IaaS, software license and support cost, and keep in mind adjustments for operation and support costs for private cloud.
   g. Understand any updates needed for IT security and audit controls (e.g, SSAE 16, ISO, FedRAMP, etc.).
   h. Evaluate and plan for staff training for the private cloud deployment.

3. Testing the workloads in the private cloud:
   a. Select deployment automation tools for infrastructure as code, and configuration management tools.
   b. Test all workload deployments and measure deployment time.
   c. Perform full performance benchmarking tests for selected workloads.
   d. Optimize infrastructure resource types based on the performance test results and avoid over-provisioning.
   e. Test your data migration strategy and procedures to understand the complexity and the time to complete the migration.
   f. Complete a few dry runs of the migrated application to test performance.
   g. Complete a security requirement evaluation for the migrated applications, then test performance again with security controls enabled.

4. Executing the migration:
   a. Engage a cross-functional steering committee (e.g. Finance, Operations, Engineering) to review the strategy and plan.
   b. Ensure proper communication and engagement are in place between the stakeholders and the operations team.
   c. Complete all the updates to IT security and audit controls (SSAE 16, ISO, FedRAMP, etc.).
   d. Develop a detailed migration plan with a schedule for each application/workload.
   e. Ensure monitoring and ticketing integration systems are in place prior to going live.
   f. Ensure end-to-end user acceptance testing (UAT) is done before final cut-over from the public cloud. Maintain the public infrastructure for at least two-to-three weeks post-migration.
   g. Ensure sufficient private cloud infrastructure capacity is in place to meet the business's service level agreements (SLAs) and to be validated by all stakeholders.
   h. Ensure continuous monitoring and optimization for the migrated application/workload.

Summary

This guide helps traditional and cloud-native application specialists prepare and execute migration plans using cloud-enabled development patterns on OpenStack clouds. OpenStack and other communities provide tooling to use the OpenStack service APIs, and package and automate application deployment. Containers, container management systems PaaS, and other new technologies are also integrated to help you use one programmable platform for all your infrastructure needs.

To avoid hidden costs and achieve proper application scaling, thoughtful considerations and detailed planning is required for public, private and hybrid cloud computing, containers, data migration, cloud-bursting use and cloud-to-cloud migrations. The checklist provides a clear path to public-to-private cloud application migration for organizations that find non-OpenStack public clouds to be expensive and inflexible.

The OpenStack community of more than 70,000 members, across over 180 countries and 650 companies is available to answer questions and offer their experience and expertise through mailing lists, Internet Relay Chat (IRC), and working groups. We always welcome new application engineers who bring new ideas to improve migration and new cloud-based development.

For more information, please visit these resources.

Resource	Link
Application development on OpenStack	*https://www.openstack.org/appdev/*
Development resources for OpenStack Cloud	*https://developer.openstack.org/*
OpenStack documentation, including the API Guide	*https://docs.openstack.org/*
Join the OpenStack community	*https://www.openstack.org/community/*
User stories	*https://www.openstack.org/user-stories/*

Resource	Link
Join the conversations on IRC	*https://wiki.openstack.org/wiki/IRC*
Community-written book, “OpenStack Cloud Application Development”	Available for purchase at **Wiley Online Library** *(http://www.wiley.com/WileyCDA/WileyTitle/productCd-1119194318.html)* or **amazon.com** *(https://www.amazon.com/OpenStack-Cloud-Application-Development-Adkins/dp/1119194318)*

Join us today. Read how other organizations use OpenStack. And enjoy the open, programmable platform for your applications.