Architecting Applications for the Cloud

To realize the full cost and agility benefits of cloud computing, enterprise customers must make an intentional shift in their approaches to application architecture. Using OpenStack* as a foundation for application development and deployment, paired with platform-as-a-service (PaaS) or container frameworks, can help accelerate adoption and the rate of application deployments to the cloud. As a result, organizations can get more value from their investments in the technology.

Businesses that deploy cloud infrastructure often have application-development teams that are slow to adopt the new infrastructure and processes. Driving up the level of cloud adoption using management directives has the risk of stifling the very innovation that cloud computing is meant to enable; it may even encourage developers to look for ways to avoid the shift to cloud development.

A better approach is to focus on engaging architects and developers, rather than placing requirements on them. Wise IT decision makers should recognize that rank-and-file technical staff may not view cloud technology as compelling in and of itself. Infrastructure engineers value the benefits of reduced total cost of ownership (CapEx and OpEx) while developers are looking for time-to-market improvements. Strategies that pursue both as a single effort can accelerate adoption and value to the business.

Through careful planning and the use of best practices it is possible to foster an environment where adoption enables the broader set of goals shared by line-of-business managers, application developers, and the organization as a whole. Success in this area can help drive results that meet the expectations enterprises have for taking advantage of their cloud investments with faster, lower-cost adoption of new services and capabilities for end users.
Current Factors Driving Application Architectures

Software teams are faced with accommodating a broad set of requirements at the same time that their budgets are being reduced. As a result, these teams are pursuing strategies that enable multiple initiatives to be achieved through a common software design model and a new set of technologies. Cloud adoption seldom emerges as a primary objective for software teams. One best practice is to use cloud deployment as a means to accelerate application delivery, thus enabling other objectives. Some of the trends that currently impact the efforts of internal application-development teams (and help shape their goals and strategies) are illustrated in Figure 1.

![Figure 1. Industry trends that are impacting the enterprise.](image)

The drive for an improved user experience has replaced the traditional focus on functionality alone to drive development goals and purchasing decisions. Users expect a seamless experience across the spectrum of devices, operating environments, browsers, and other factors. Those requirements must be met in the face of disruptive influences such as the ongoing rise of social media and public cloud apps. Management typically places additional demands on these efforts, in terms of faster time to market, lower cost, and increased interoperability with other systems. In addition, the landscape of cyber-security threats continues to become more hazardous and complex.

Intel’s Open Source Technology Center (OTC) has an ongoing commitment to the community, in terms of both enabling OpenStack for Intel® platform features and helping advance best practices for the use of open source software in application deployments. Together, these goals help enterprise organizations realize the full value of cloud computing. Sharing the outcomes of OTC’s efforts in this paper—including best practices established with OpenStack adoption internally at Intel—is part of that effort. The strategy advocated by OTC to enable application architectures for the cloud can be summarized as follows:

- **Transition the front end to a multi-platform orientation.** Where standardizing on a single platform used to drive efficiencies, the need to support large numbers of client platforms has caused that reality to shift, especially as the “bring your own device” trend continues to grow in influence.

- **Transition the back end to the cloud.** Multi-tenant, auto-scaleable, distributed resources offer dramatic efficiencies and the ability to expose functionality to multi-platform clients as services using platform-as-a-service (PaaS), infrastructure-as-a-service (IaaS), and software-as-a-service (SaaS) as broadly as possible.

  This approach continues to extend new benefits in terms of an enhanced, flexible user experience and agility for the business. At the same time, it accelerates the ability of the development organization to deploy new approaches, libraries, frameworks, and tools that facilitate further advances.

Fulfilling Expectations Surrounding Cloud Adoption

Most broadly, IT decision makers who have implemented OpenStack or other private cloud platforms typically want to see applications architected for cloud to justify the investment. Looking slightly deeper, however, their interest tends to be focused on time to market and end-user productivity. As mentioned above, user experience has become central, including the ability to deliver intuitive applications that take advantage of the capabilities of multiple client devices and can embrace new capabilities as they emerge, such as voice, touch, and gesture interactions.
Organization-Level Benefits

At the organizational level, successful adoption of private cloud can enhance agility, efficiency, and security, aligning the needs of developers with the rest of the organization, as shown in Figure 2.

Figure 2. Enabling the business with private cloud.

The more agile infrastructure and measures to automate workflows dramatically reduce time to market for enterprise apps and databases, cutting deployment time frames from weeks to just minutes in many cases. The shift from single tenancy to multi-tenancy creates higher compute density, and that smaller environment can be supported by fewer personnel, reducing support costs. Because apps and data are centralized, they reside on infrastructure that is under the control of the IT organization, rather than business units or third-party providers, which enhances the ability to follow coherent, standardized security policies.

Self-Service for Developers

Relieving developers from having to be concerned about system engineering concerns is a proven strategy to accelerate time to market. To that end, Intel has built a robust, flexible self-service environment for developers atop the OpenStack cloud operating environment. Because individual needs and preferences vary broadly, this environment is accessible using a variety of options that range from command-line interfaces to a web-based UI.

Using any of these options, application teams can easily specify the CPU, memory, disk, and other resources required, build databases, and upload and deploy applications with a streamlined process. In addition to typical application platform services associated with capabilities such as analytics, messaging, data, and web resources, the self-serve environment offers a number of additional, underlying services. These support such capabilities as monitoring, user interface and API components, vulnerability scanning, and single sign-on (SSO).

These tools make using the cloud environment more attractive for developers, accelerate time to market, reduce costs, and streamline operations for everyone involved (including the system engineering organization). Beyond those distributed enabling benefits, self-service supports broader goals and initiatives that range from adoption of private cloud and PaaS to facilitating software-defined networking, manageability, and security. Philosophically, the goal of this approach meshes with Intel’s philosophy of directing development practices through constructive engagement instead of by issuing mandates.

Architecture Design Patterns

The different manifestations of multi-tier architecture on conventional (non-virtualized), virtualized, and cloud infrastructures illustrates many of the design considerations involved in architecting applications for the cloud. This section introduces the variations in a classic, three-tier architecture as they relate to these different deployment environments. A three-tier architecture, as illustrated in Figure 3, distributes functionality as follows:

- **Data tier** provides the data-storage back end for the application.
- **Application tier** handles requests from the client front end and implements business logic.
- **Web (presentation) tier** provides the user interface that the client interacts with.
The architecture shown in Figure 3 incorporates load balancers to improve performance and reliability, as well as firewalls to enforce data-interchange rules between the tiers. In a conventional infrastructure, each tier of the architecture is typically deployed on dedicated host hardware. The systems in each tier are statically configured with the hostnames of the other hosts they interact with to implement the application logic.

The key advantage of this architectural approach is that each tier can be developed and operated independently of the others, as long as the interfaces and data models of each remain compatible. This also facilitates separation of duties within larger organizations for audit, compliance, and related purposes.

This model assumes that services are developed as a set and deployed statically. Dependencies and data connectivity between the tiers are captured as set values in configuration files. For example, an email client would be configured with the host name or IP address of the mail server, rather than dynamically discovering the nearest mail server or the one with headroom available to satisfy a given service request. Scaling of capacity in this environment is manual in nature, either scaling up by adding physical resources to the server or scaling out by adding a new server and changing configuration settings elsewhere in the environment.

Virtualization incrementally changes the implementation topology of multi-tier architectures. Rather than being deployed on dedicated hardware, components within the tiers are deployed on virtual machines (VMs) that are decoupled from specific physical systems. This approach enables dynamic scaling of each tier independently by spawning new VMs on demand as well as enhanced agility and reduced costs through accelerated provisioning and automation. Unlike cloud environments, however, virtualized architectures typically use the same static configuration models as non-virtualized ones.

Mapping Three-Tier Architecture to OpenStack
Implementing a three-tier application architecture on a private OpenStack cloud is analogous to doing so on a conventional or virtualized server infrastructure, as illustrated in Figure 4. In this sample topology, Neutron’s load-balancing-as-a-service (LBaaS) functionality distributes requests among the active servers in the web and application tiers. This function can be accomplished using any of several methods, including round robin, least connections, or random placement. Security groups perform port-based filtering between each tier, analogous to the firewall functionality in Figure 3.

![Figure 3. A typical three-tier architecture.](image1)

![Figure 4. Three-tier architecture mapped to OpenStack*.](image2)
Decomposition into APIs and Micro-Services

One of the most significant advantages of using cloud to deploy the multi-tier architecture is that multi-tenancy provides the ability to scale each tier horizontally when resource demand requires it. Auto-scaling can be implemented in both the web and application tiers, based on monitoring of the utilization levels on active nodes by means of Ceilometer, OpenStack’s orchestration and telemetry service.

When the load on these nodes reaches preset levels, Ceilometer generates an alert that triggers orchestration processes based on the Heat service. Heat orchestration is discussed in greater detail below. A number of metering options are available using Ceilometer, based on factors such as processor, memory, network, and disk I/O utilization. In addition to triggering events in response to high utilization, similar behaviors can be initiated when activity levels are especially low, to save money by disengaging unneeded resources.

Cloud Raises the Bar for Sound Multi-Tier Design Practice

While most development organizations embrace the convention of multi-tiered application architectures, it is also common to be somewhat lax about keeping the layers completely separate. As a matter of convenience, developers may incorporate some presentation logic into the application tier, for example. While such practices often cause little or no trouble, they can also lead to issues as applications evolve and become more complex over time.

Cloud modalities inherently enforce some aspects of multi-layer design and stateless architecture, and in some other cases, enforcement can be automated as a matter of choice. In either case, moving to the cloud can be used as a means to help foster sound architectural practices.

Taking advantage of cloud infrastructure has significant implications for application architecture and design, as summarized in Table 1. To effectively make use of cloud capabilities, architects must place elasticity, self-service, and multi-tenancy as primary considerations. In practice, this requires a shift from a monolithic, centralized design with synchronous operation based on tightly coupled infrastructure to one that assumes distributed, asynchronous capabilities on shared-nothing architecture.

Table 1. Features of conventional versus cloud-aware application architectures.

<table>
<thead>
<tr>
<th>Conventional</th>
<th>Cloud-Aware</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Monolithic</td>
<td>• Distributed</td>
</tr>
<tr>
<td>• Centralized state</td>
<td>• Micro-services</td>
</tr>
<tr>
<td>• Synchronous</td>
<td>• Asynchronous</td>
</tr>
<tr>
<td>• Single-tenant</td>
<td>• Multi-tenant</td>
</tr>
<tr>
<td>• Tightly coupled</td>
<td>• Shared-nothing</td>
</tr>
</tbody>
</table>

Decomposition into APIs and Micro-Services

A core characteristic of cloud environments is that instances of services are more frequently initiated, relocated, or discontinued. These changes may result from factors such as workload fluctuations, software failures in service instances, or infrastructure failures.

The micro-service design pattern provides resiliency by functionally decomposing monolithic services into more fine-grained components, each of which has a single function. For example, an email messaging system can be broken down into separate API methods for creating, reading, updating, and deleting messages. Using discrete micro-services in this manner has a number of advantages that include increasing elasticity, performance, reliability, and ease of deployment.

In particular, decomposing functionality into micro-services simplifies development and deployment of software updates. A software developer can own an entire micro-service and can develop and release updates to the service independently from other developers. For example, a developer who creates an improved message caching feature can release it to production without integrating with the other message-store functionality. This approach decouples the development of individual features from others, so developers can work in parallel and roll out updates on their own schedules. This capability explicitly enables DevOps initiatives.

Another benefit of micro-services is that if a failure occurs, it is relatively easy to identify the source of the problem because the scope is bounded by the micro-service at fault. Therefore, developers can identify which update triggered the failure and determine the micro-service that must be analyzed during troubleshooting.
Heat for Auto-Scaling

Heat is the OpenStack orchestration project. Heat templates are text files that describe infrastructure and resources for cloud applications, including entities such as servers, floating IPs, volumes, security groups, and users. The Heat auto-scaling service can set up resource-monitoring alarms in conjunction with Ceilometer that can be used to spawn and configure web, application, and database VMs to control resources. The following code snippet spawns web servers and passes in commands to perform updates and installations of Apache.*

```yaml
web_server:
  type: OS::Nova::Server
  properties:
    name: { get_param: server_name }
    image: { get_param: image_id }
    flavor: { get_param: instance_flavor }
    key_name: { get_param: ssh_key_name }
    metadata: { get_param: metadata }
  networks:
    - network: { get_param: private_network_id }
  security_groups:
    - { get_param: security_group }
  user_data: |
    #!/bin/bash -v
    sudo apt-get update
    #Install Apache package
    sudo apt-get install apache2
    sudo /etc/init.d/apache2 restart

web_autoscale_group:
  type: OS::Heat::AutoScalingGroup
  properties:
    desired_capacity: 2
    min_size: 1
    max_size: 5
  resource:
    type: lib/heat_web_tier.yaml
    properties:
      ssh_key_name:
        get_param: ssh_key_name
      server_name:
        get_param: web_server_name
      instance_flavor:
        get_param: web_instance_flavor
      image_id:
        get_param: image_id
      private_network_id: {get_attr: [network_setup, web_private_network_id]}
      security_group: {get_attr: [network_setup, web_security_group_id]}
      pool_name: {get_attr: [network_setup, web_lbaas_pool_name]}
      metadata: {"metering.autoscale_group_name": "web_autoscale_group"}

web_scaleup_policy:
  type: OS::Heat::ScalingPolicy
  properties:
    adjustment_type: change_in_capacity
    auto_scaling_group_id: { get_resource: web_autoscale_group }
    cooldown: 240
    scaling_adjustment: 1

web_scaledown_policy:
  type: OS::Heat::ScalingPolicy
  properties:
    adjustment_type: change_in_capacity
    auto_scaling_group_id: { get_resource: web_autoscale_group }
    cooldown: 240
    scaling_adjustment: -1
```
Trove for DBaaS

Trove provides database-as-a-service (DBaaS) for OpenStack that provides the features of a relational or non-relational database without having to handle complex administrative tasks. DBaaS enables developers to provision and manage multiple, isolated database instances as needed, using their database platform of choice, including MySQL*, MariaDB*, PostgreSQL*, MongoDB*, and others. Trove offers the following capabilities and advantages:

- **Simplicity** from multiple database platforms in easy-to-use, prepackaged configurations to meet diverse developer needs with a single service.
- **Agility** through rapid provisioning in conformance with corporate standards and guidelines without having to delve into administrative details.
- **Innovation** enabled by access to emerging database technologies for performance, scalability, and availability within corporate guidelines, without needing to be a database specialist.

IT organizations and cloud service providers can configure Trove to provide a variety of database configurations in the form of preconfigured “guest images.” Guest images encapsulate database administration best practices, helping ensure that the system is operated and managed properly without requiring the IT staff to be experts on every database technology. Guest images can be configured by the operator or downloaded from publicly available sources.

Best Practices

The journey at Intel engaging developers to adapt cloud as their platform of choice has revealed a number of technology approaches that can be regarded as best practices for other organizations as they take on similar efforts. A few of these practices are described in this section.

Cross-Origin Resource Sharing (CORS)

In support of the effort to create web applications that are fully multi-platform, many organizations focus on standard technologies such as HTML5, CSS, and JavaScript* for UI development. This is commonly regarded as a best practice in lieu of using proprietary technologies to render content on the server and then pushing HTML to the browser. On the other hand, if the browser makes a request to a service on an external domain and owned by a third party, issues can arise if that service is not CORS-enabled. CORS is a mechanism that enables web pages to request external, restricted resources from other domains. Because developers cannot control whether such external services are CORS-enabled, best practices call for writing a wrapper to call legacy services from the service layer, rather than directly from the browser.

NTLM Token Translation Architecture

Another issue that the cloud-architecture team at Intel encountered concerned user authentication. By convention, applications at Intel use SSO based on NT LAN Manager* (NTLM), which is a Microsoft Windows*-dependent suite of technologies. Because the cloud infrastructure is not Windows-based, cloud applications lacked an SSO mechanism, meaning that users would need to repeatedly authenticate as they accessed multiple resources. The resulting non-optimal user experience could cause some developers to hesitate to write applications for the cloud.

The solution was to create a token translation architecture that handles authentication through a series of HTTP redirects, as illustrated in Figure 5. This mechanism requests an authentication token from the Windows SSO service on the initial request and then allows subsequent calls to be handled using sessions, which reduces the performance impact. The steps illustrated in the figure are as follows:

1. Service call (initial).
2. Redirect to get SSO token.
3. Request SSO token from SSO service (Windows authentication).
4. SSO service sends SSO token.
5. Resend request with SSO token.
6. SSO token verification request.
7. SSO token verified.
8. Service response.

From the user’s perspective, this approach provides transparent SSO, mitigating a shortcoming that could otherwise interfere with developer adoption of the internal cloud infrastructure.
Forward-Looking Technology Adoption

Similar to the NTLM token translation architecture discussed above, teams in charge of private cloud architectures must enable the environment for emerging technologies, setting the stage for adoption by internal development teams. One example of current interest at Intel is to enhance PaaS offerings by pursuing containers. This effort seeks to retain the current benefits of agility, time to market, and cost, while also addressing unique configurations and hyperscale.

A key limitation at present is that while most enterprise applications run on standard configurations in the new web architectures, container-based approaches typically don’t provide automatic template capabilities, and manual configuration is required. Ongoing work in this area seeks to automate template-based configuration, which promises to make container-based topologies more efficient and viable, even for specialized usages.

Initial Results

Intel IT has seen promising results in adopting this approach internally. Application deployment times have fallen from months with traditional dedicated systems to weeks for IaaS, and even hours for PaaS and DBaaS deployments. Meanwhile internal benchmarks show that a private cloud can be cost effective when compared to a public cloud, with TCO savings of around 50 percent for the applications that Intel IT studied. This combination has proven popular with internal developers, with steady increases in developer adoption and application deployment over the course of 2016, as shown in Figure 6.

![Figure 6. Internal adoption of private cloud at Intel.](image-url)
Conclusion

Successful implementation of private cloud technology such as OpenStack can effectively reduce time to market for internal applications from weeks to days or even hours. The key to success with this approach is to drive adoption by engaging with internal development teams. Rather than managing the process by mandate, enabling teams by means of self-service resources and mechanisms to streamline deployment are best practices that have proven to engender success.

Organizations can align private cloud adoption with developer goals such as improving the user experience, supporting multiple device platforms, and cutting time to market. By doing so, they can accelerate the path to realizing the full potential in terms of agility, efficiency, and cost savings from internal applications, helping derive the full value from investments in private cloud.

Learn more:
intel.com/openstack