IT@Intel White Paper

Intel IT
IT Best Practices
Cloud-Aware Applications
May 2013

Maximizing Cloud Advantages through Cloud-Aware Applications

Intel IT is helping Intel developers design cloud-aware applications that maximize cloud advantages such as self-service provisioning, elasticity, run-anywhere design, multi-tenancy, and design for failure.

Catherine Spence
Enterprise Architect, Intel IT

Munir Ghamrawi
Cloud Automation Engineer, Intel IT

Ravi Giri
Staff Engineer, Intel IT

Winson Chan
Cloud Automation Engineer, Intel IT

Esteban Gutierrez
Senior Information Security Technologist, Intel IT

Das Kamhout
Principal Engineer/Cloud Lead, Intel IT

Executive Overview

As Intel IT continues widespread adoption of cloud computing, Intel software development is undergoing a major shift. Intel architects and developers are learning to design cloud-aware applications that maximize cloud advantages, such as self-service provisioning, elasticity, run-anywhere design, multi-tenancy, and design for failure.

Giving developers the tools they need to build cloud-aware applications benefits Intel several ways.

- Faster time to market for enterprise applications
- Rapid adoption of cutting-edge technologies
- Cost and resource efficiencies through elastic, on-demand cloud infrastructure
- Greater flexibility in hosting applications using a hybrid cloud model
- New innovation opportunities through easier data integration
- Easier implementation of new mobile computing devices and usage models

Designed from the ground up for cloud deployment, cloud-aware applications require a different way of thinking. To speed implementation, Intel IT is applying key learnings from grid computing, as well as adopting best cloud practices such as consumable web services and high-availability designs. Our strategy is to architect enterprise applications with a cloud back-end and multiplatform front-end, actively exposing and consuming web services offering built-in security. Through one-day code-a-thons and other events, we're training developers in this new paradigm and validating our strategic agility target of innovative idea to production in one day.

By increasing interoperability between private and public clouds, the design of cloud-aware applications are an important step in our progression toward a federated, interoperable, and open cloud as our standard way of providing services. We believe that perfecting cloud-aware application development is a vital step to maximizing hybrid cloud advantages and bolstering the reliability, security, and agility of our enterprise applications.

Contents

IT@INTEL

The IT@Intel program connects IT professionals around the world with their peers inside our organization – sharing lessons learned, methods and strategies. Our goal is simple: Share Intel IT best practices that create business value and make IT a competitive advantage. Visit us today at www.intel.com/IT or contact your local Intel representative if you'd like to learn more.

BACKGROUND

Intel IT began a transition to cloud computing in 2010 and continues making substantial progress toward our goal of a hybrid cloud hosting model. These efforts are enabling us to achieve high levels of agility, scalability, and efficiency. While our initial focus centered on optimizing data center and cloud infrastructure, software applications play an essential role in how, when, and where IT services are consumed. We recognize that to obtain even greater business benefits, we need to encourage the development of cloud-aware applications designed from the ground up to take full advantage of the cloud.

To help understand the adoption of cloud computing in the enterprise, Intel uses the Cloud Maturity Model (see Figure 1) provided by the Open Data Center Alliance (ODCA). This coalition includes more than 300 leading businesses that together represent billions of dollars in annual IT investment, cloud research, and projects.

The Cloud Maturity Model provides an end-to-end visualization of how cloud use in the enterprise will evolve over time. As cloud implementation matures, it becomes more sophisticated, comprehensive, and optimized. Based on ODCA industry experience, many large enterprises are progressing along the same overall trajectory but at different rates of adoption.

The time ranges indicated for each stage in Figure 1 are specifically for Intel IT. For example, we were at version 1.0 in the 2010-2011 time frame. By 2015, we are projecting to be at version 3.0, offering a federated, interoperable, and open cloud spanning all cloud service models—software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS). We will be able to **federate** user identities among private and public clouds. Our workloads will be able to **interoperate** across clouds easily with minimal switching costs. And we are already building our next-generation enterprise private cloud utilizing **open** solutions to facilitate a fast introduction of capabilities based on the open source communities cadence. With a federated, interoperable, and open cloud, Intel IT will have the flexibility to decide where to rapidly source cloud services based on user demand, cost, location, and regulatory requirements.

To build on our progress and to unlock the full value of the cloud, Intel IT is now training developers to design cloud-aware applications that take advantage of our PaaS and IaaS offerings. Cloud-aware applications improve the ability to consume cloud services, taking better advantage of infrastructure that can grow and shrink according to demand. These applications also seamlessly move on and off premises, providing the ability to optimize location, cost, capabilities, and risk. Compared to traditional applications, cloud-aware applications can be more resilient and can enable the delivery of higher levels of services, such as a faster mean-time-to-recovery (MTTR).

Cloud-aware application development techniques build on best practices of traditional application development and can be hosted more quickly in the enterprise private cloud through self-service tools and automation. Intel IT is at the point where developers can choose either of the following:

- **PaaS,** which provides a pre-provisioned environment with operating system, abstracted middleware, and infrastructure that allows developers to rapidly deploy applications without having to provision servers. Our PaaS implementation enables rapid application deployment through self-service, on-demand tools, resources, automation, and a hosted platform runtime container in the enterprise private cloud. PaaS facilitates the creation of cloud-aware applications through the use of templates, resource sharing, reusable web services, and large-scale multi-tenancy.

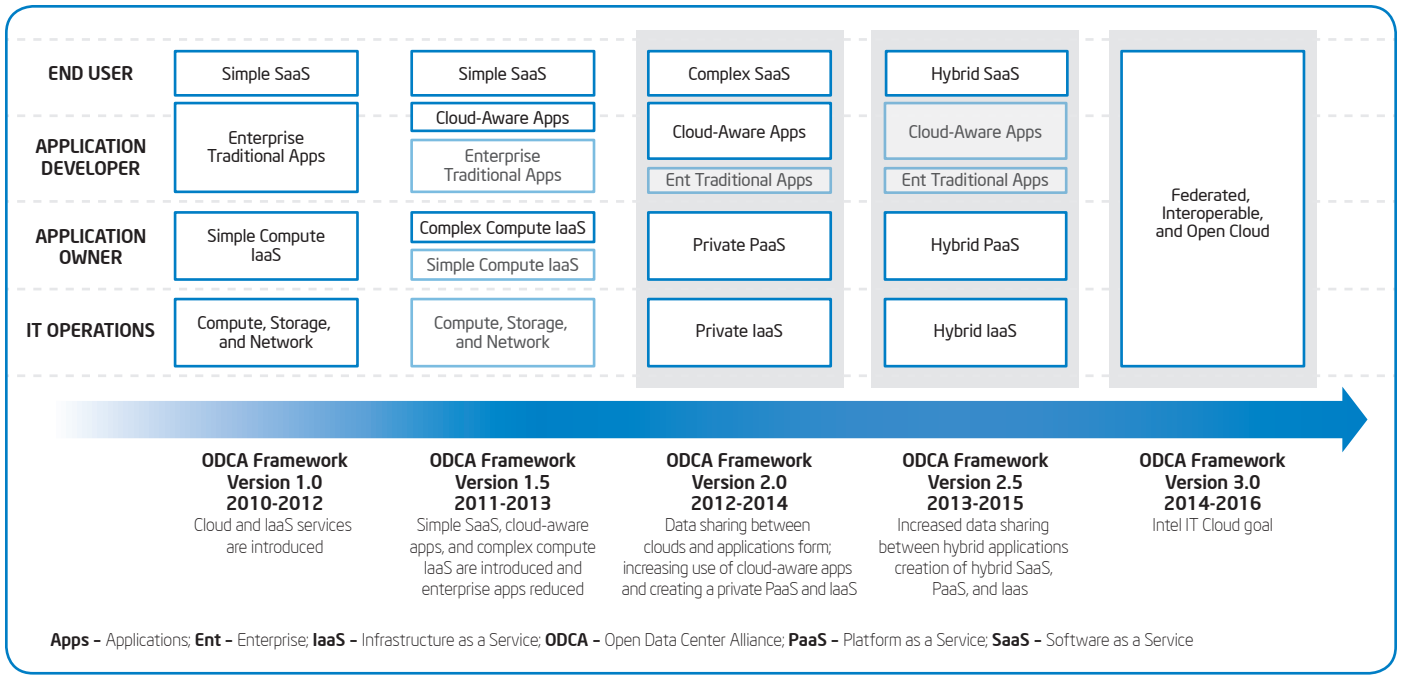

Figure 1. Using the Open Data Center Alliance Cloud Maturity Model is helping Intel IT progress to a federated, interoperable, and open cloud—the ultimate goal recommended by industry organizations.

- IaaS, which supplies developers more control when needed over the entire application stack or when the application requires a level of isolation from other applications on the same application stack. In addition, our IaaS implementation exposes infrastructure services as APIs and provides a user interface that makes it easy for developers to consume compute, network, and storage. Developers choose IaaS for more complex applications or for applications with requirements for very high levels of availability.

As developers start to regularly develop cloud-aware applications, Intel will be able to more rapidly and readily adopt and apply cutting-edge solutions and achieve faster time to market. Instead of waiting for physical infrastructure or custom, internally developed services, developers will be able to consume services as soon as they need them.

BUSINESS CHALLENGE

Organizations face several challenges in making the transition to cloud-aware application development. Having already invested considerably in developing and maintaining legacy applications, organizations may be resistant to change. Cloud-aware application development also requires a new way of thinking—developers must be retrained to think in all ways cloud.

Many organizations have already completed the first stage in the Cloud Maturity Model, adopting virtualization to consolidate workloads previously hosted on dedicated physical servers. This transition involves little or no change in the applications themselves, which are still written for a physical environment and have traditional expectations and dependencies.

- High availability derived from infrastructure designed at a cost for high resilience through a low mean-time-between-failure (MTBF).
- Isolation achieved through "owning" a specific server—or virtual machine (VM)—and database. These types of dedicated resources significantly minimize potential contention issues or conflict with other applications, particularly in virtualized servers.
- Security assumed from placement of applications behind the firewall.
- A stationary location accompanied by a set of IP addresses and server names that do not change.

Cloud-aware applications change these expectations to take best advantage of what makes the cloud so fluid and elastic. Consider how much different the thinking is for cloud-aware applications.

- High availability achieved by software systems designed for fast MTTR instead of low MTBF.
- Resource sharing among applications instead of isolation.

- Security derived not from a firewall but from the combination of a secure cloud service foundation, security features built into the application, and enterprise security capabilities delivered as consumable services.
- A run-anywhere design where, instead of a stationary location, individual workloads move in and among private and public clouds and across the world based on demand and policy.

Cloud-aware applications offer additional benefits, including the following:

- A design that supports multiple tenants (groups of users) and common instances of applications, platforms, and infrastructure.
- Visibility and control limits that keep individual applications from having full control over the infrastructure and operating system to make changes to abstracted configuration settings.
- Self-service capabilities that make all cloud aspects available to the user without creating IT service requests.
- Elasticity features that allow applications to grow and shrink on demand to optimize pay-for-what-you-use cloud pricing models.
- Evolvable characteristics in which both software and hardware components may update underneath an application.
- A composable design in which applications are divided into discrete services for use alone or for fast reuse in combination with other services to create more complex applications.

For most developers, cloud-aware applications are a major paradigm shift. Making this shift even more important to embrace is the proliferation of different mobile and bring-your-own (BYO) devices throughout the enterprise. It is increasingly necessary to decouple applications from clients to avoid the time and expense of dealing separately with each form factor, browser, and operating system. The cloud provides a perfect back-end that can scale to suit the growing demands from the explosion of devices, as well as allow software developers to build applications and services that enrich the lives of IT consumers with always-on, anytime, anywhere access.

SOLUTION

To accelerate the organization's ability to develop cloud-aware applications, Intel IT focused on three key areas: applying key learnings from grid computing, developing consumable web services for use as software components, and designing for high availability. Through these efforts we have been able to decouple applications from clients, enable easy reuse of software components, provide security at every layer, and design for failure.

Applying Key Learnings from Grid Computing

The concept of grid computing predates that of cloud computing and provides a series of best practices for designing cloud-aware applications. In grid computing, individual computers in multiple locations are connected through a dedicated, high-speed network to share memory, storage, and other resources to work on a common goal. These computing resources are coordinated by a resource or workload scheduler that uses advanced algorithms to decide what kind of computing should be done on which compute node and at which location. This distributed computation is similar to many cloud solutions that have the elements of geographical distribution, workload scheduling on VMs, and a goal of maintaining high utilization.

Since 2005 Intel has had a global grid computing capability that enables jobs—discrete computing workloads—to run on up to tens of thousands of compute nodes anywhere in the world. Many characteristics of grid computing apply within a cloud computing environment.

- **Run anywhere.** Intel's global grid for silicon design computing spans more than 30 geographic locations. Each project requires some preliminary work to ensure availability of required data, metadata, and configurations.
- **Elastic scaling.** Elastic scaling has enabled the use of idle resources around the world to meet millions of hours of unforecasted computing need or to reduce capital expenditure for forecasted needs by using idle capacity. In the last 12 months, in just the top three data center locations at any given point in time, this capability enabled an average of >87,000 workloads (normalized to adjust for server platform throughput improvement) to run.
- **Multi-tenant resource sharing.** Our grid environment focuses on maintaining high utilization, a state which is achieved by allowing applications across projects and business units to share a common pool of computing resources.
- **Design for failure or stateless computing.** In stateless computing, software does not track configuration settings, transaction information, or any other data for the next session. This approach makes it perfect for grid computing where, from an application point of view, every compute node in the grid is exactly the same as another. Failure of an individual compute node or nodes simply involves rescheduling a particular application instance on another compute node and is completely transparent to the end user.

Moving from a traditional localized cluster to a global grid involves many changes to the methods adopted by application developers. There can be no dependencies on local environments, such as references to specific machines or IP addresses to data paths.

Intel IT created a framework to allow developers to migrate existing applications and tools to

run in a global grid environment. An abstraction layer, or virtual pool, allows the resource scheduler to coordinate resources across many locations. A global namespace ensures data paths are always consistent irrespective of the geographic location in which the application is running. And a global configuration management system ensures that the compute nodes that the resource scheduler chooses to run a given application are consistent and optimized for that application.

SIMILARITIES AND DIFFERENCES BETWEEN GRID AND CLOUD COMPUTING

As a precursor to cloud computing, grid computing provides some guidance in developing cloud-aware applications. Grid computing, like cloud computing, is event-driven and relies heavily on queuing. The need to disassociate the application from dependencies on a specific system, environment, or data center, involves common techniques for both. Other common characteristics include a scale-out versus a scale-up environment. In grid computing individual compute nodes are relatively small, but there are large numbers of them. In cloud computing, compute nodes may be both large and numerous.

A key difference between grid and cloud computing is elasticity. Cloud services can grow and shrink dynamically to meet demand, whereas grids are static environments. In addition, the cloud provides the ability to measure consumption and resource usage, as well as provides self-service for on-demand resources.

For developers, most grid computing environments by design offer a homogeneous base of common operating systems, software libraries, and more, which simplify application development. A cloud computing environment, particularly a hybrid cloud, may present a mix of environments. Another key difference is that almost all applications using grid computing run in batch mode and thus involve no user interaction. In contrast, a significant number of applications in cloud computing involve user interaction.

ADOPTING PRACTICES FROM CLOUD COMPUTING

While Intel IT still runs a large design grid today, we expect to see an evolution to the cloud. Moving the grid to the cloud will give us even better resource utilization as we consolidate environments, enabling us to direct workloads to specific processors that support high-performance computing.

Intel IT has embarked on a cloud design program to transform the grid environment to incorporate several cloud computing elements.

- **Self-service.** The ability to specify required configurations, metadata, and avoid other preliminary work involved in running applications globally offers a significant benefit to grid computing.
- **Visibility and control.** We are investing in capabilities that allow end users complete visibility to the state of the environment and the ability to manage resources either through user interfaces or standardized web service APIs.
- **Composable.** This key cloud characteristic is expected to bring significant benefits to grid computing. For example, some applications that run in grid environments need network awareness to ensure dependent jobs land on compute nodes that are nearby. In the future, we expect to go from network awareness to network control where the application can define what kind of network configuration is required for optimal execution.

While Intel IT's grid computing experiences significantly influence our cloud journey, and in return our cloud experiences provide new insights to our grid computing efforts, we foresee a future when these environments and usage patterns are supported by a single federated, open, and hybrid cloud.

Developing Consumable Web Services

A web service is a software function provided at a network address over the web or the cloud. As Intel IT continues implementing an open cloud, we are driving the creation of consumable web services at every layer. Offering consumable web services is important because it is the only way we can massively automate the cloud and enable self-service. Likewise, our software application developers need to adopt the view that in the future a software developer will want to mash-up their software application with others to create new business processes and automation at scale.

Intel's preference for web services uses a representational state transfer (REST) model. RESTful APIs rely on HTTP and do not require XML-based web service protocols, such as SOAP or WSDL, to support their lightweight interfaces. Other HTTP advantages include the ability to be called from any platform, efficiently cache requests, and easily scale out. The RESTful model enables developers to quickly reuse application capabilities through standardized methods such as post, get, put, and delete. These standardized methods reduce the learning curve associated with consuming a new software service and shield the developer from the underlying technology implementations.

Using open source web services for more utilitarian functions helps us accelerate development and save coding time for custom tasks, such as connecting to the back-end. More and more web applications have published APIs that enable software developers to easily integrate data and functions instead of building them personally. Easy-to-use mashup composition tools are also emerging. In addition, we encourage Intel developers to expose their applications through web services so other developers can combine multiple applications into

Intel IT Consumable Web Services Guidelines

All software products and infrastructure services should include a consumable API (web services) to automate and integrate the capabilities within more complex business workflows. This applies to the following types of software:

- Software acquired by Intel and hosted at Intel to construct an enterprise private cloud
- Software hosted by a cloud provider and consumed by Intel in either a standalone or hybrid cloud model
- Software services internally developed by Intel
- Infrastructure services running at Intel or consumed by Intel, including all hardware solutions that IT developers consume, regardless of their location or their authorization to use it

We expect all web services to be written in a manner consistent with good API design and cloud computing practices.

- Functionality provided through a command line interface (CLI) or graphical user interface (GUI) must also be available through an API. Furthermore, CLIs and GUIs are implemented as wrappers for public APIs instead of relying on private APIs or other mechanisms not available to application developers.
- APIs are discoverable at runtime with clear service descriptions, versioning, and published service-level agreements. Ideally, these are exposed through a service catalog.
- Programming examples are made available and in commonly used languages. Developers are encouraged to provide client libraries (bindings, proxies) for various commonly used languages to accelerate adoption.
- We encourage use of the representational state transfer (REST)-based architecture model, including conventional uniform resource identifier (URI) design, and standard HTTP methods (Post, Get, Put, Delete) and security (encrypted Secure Sockets Layer, Open Authorization). APIs are logically layered to provide useful abstraction from the core capabilities of the software and simplify their usage.
- APIs support role-based access with proper authentication, authorization, and accounting controls in place. Resources being managed by an API are further restricted to allow access to only appropriate owners and approved delegates. Operations are logged such that the system can be reviewed to identify who accessed what and when.
- Developers can select from a list of available output data types, such as JavaScript* Object Notation (JSON), XML, YAML, and HTML. We prefer the JSON format.
- APIs do not limit potential cloud users by requiring special licenses or through cost limitations on the use of the interface.
- There is a loose association between the client application and web API where calls do not depend on the prior API call (stateless compute), all state data is returned to the client, and all dynamic memory is released.
- Web services provide mechanisms to support asynchronous call back, such as publishing results to the message queue where the consumer is listening, email notification, or other forms of call back, for long-running tasks. The task is identified by a globally unique identifier. Consumers can query for status or progress of the task using this identifier.
- All application data, including cache and log data, are stored outside individual compute instances utilized by the cloud software. If there is a failure in the back-end, components resume transaction processing from any compute instance without losing any transactions.
- In general, data processing within the web service back-end is designed for eventual consistency rather than relying on real-time, synchronous mirroring to keep each compute instance in lock step. The exception is niche areas where fixed resources are involved and constant consistency is required.
- Web services are compartmentalized and capable of rapid elasticity at a massive scale. Instead of failing requests when capacity is low, we use a rate-limiting technique to throttle requests initially. When resources are completely consumed, an error is returned. We design our web services with denial of service attacks in mind.
- Web services are designed to be accessed through meaningful and easily remembered vanity names in combination with global load balancers with appropriate routing policies, such as proximity, round robin, and others. We prefer an internal root tree for APIs.
- Running web services do not rely on static TCP/IP addresses; instead they comprehend domain names.
- Web services are designed for failure, such as anticipating infrastructure failures and maintaining uptime across multiple availability zones, preferably through an active/active model.
- Response status codes conform to standard HTTP format, as maintained by the Internet Assigned Numbers Authority (IANA). The body of the error response message should include additional human-readable diagnostic information to help debug applications.
- Maintenance and ongoing development of the APIs are nonintrusive (backward compatible), enabling sustained operations without impact or requirement for downtime. Incompatible changes are addressed through API versioning so that the old and new versions are available simultaneously.

mashups spawning new applications. This practice is particularly valuable for application development throughout the business process chain. Open APIs and data sources enable fast, easy integration to produce enriched solutions for new and evolving business conditions.

Some of our latest APIs enable developers to create cloud-based applications that can detect local client device capabilities. Such client awareness enables more effective application deployment across a wide range of devices. For instance, a network bandwidth API can detect wireless signal strength to enable an application to adjust content quality or other parameters. A processor performance API can detect the CPU type to enable running content or applications that can be optimally executed and displayed on the device. A third API monitors battery life to provide periodic power-level updates that an application can use to inform a user to conclude a process or take action before a power interruption.

Designing for High Availability

Although the goal for a cloud infrastructure might be zero failure, in reality components fail, data centers shut down, services become unavailable, and latencies increase intermittently. Consequently, we design our cloud-aware applications to function even when outages occur. Designing applications to survive failures in a distributed environment affects decisions throughout the architecture, but are especially important to consider in relation to the user experience. We design our applications so they gracefully degrade and recover when services are unavailable. Intel IT encourages Intel developers to use a variety of methodologies to architect cloud-aware applications for zero downtime.

Cloud-aware application design assumes that all components including static contents and their data containers are available at all times and in reasonable proximity to the end user based on performance requirements. When a component fails, the process of pointing the application to an alternative synchronized component should be completely seamless with minimal impact to the end user. That requires the application to distribute data in multiple locations, providing complete redundancy for every component and automating the failover process.

The techniques we use for high-availability design include the following:

- Automated and repeatable deployment for application resources such as infrastructure, security, code or binary, application configuration, and data.
- Additional workloads that can be easily distributed through adding new compute nodes. This requires monitoring and logic to determine when to automatically scale out. In addition, the scaling function needs to perform rapidly to address sudden increases in demand. When demand decreases, we need to rapidly scale back as well.
- Resilience through monitoring and self-healing, such as the ability to remove bad nodes from a load balancing pool and deploy a new node to replace the bad one. The goal is to minimize downtime experienced by the end users of the application.

DESIGN FOR FAILURE

Since clouds change over time with capacity addition and removal, developers must assume parts of the cloud may occasionally fail and must actively test applications for their ability to handle and respond to such failures. We encourage Intel developers to test resiliency by randomly shutting down parts of their environment.

To design for failure, we encourage developers to think about failure from the application's viewpoint. This means focusing on the internal architecture of the application, as well as ensuring proper implementation across multiple distributed locations and availability zones. When an application is designed with multiple modules that can call each other through an API, any module failure can recover independently in less time than if the application is just one big component. Componentizing can be accomplished through application sharding, data sharding, multiple deployments (instances) of the same components, and utilizing multiple regions for the same deployment. Another technique is to consider how an application can provide a degraded but available level of capability when certain parts of the application are experiencing an outage. For example, if a recommendation engine is experiencing an outage, the end user should still be able to perform transactions in the system, only without the recommendations.

USE STATELESS COMPUTING

In traditional applications, state is usually stored in memory. However, state stored in memory can be lost if there is an outage. It is also difficult to share state among multiple application instances. Therefore, for cloud applications we find that stateless applications and protocols work best for our needs.

Stateless protocols such as IP and HTTP treat each request as an independent transaction that is unrelated to any previous request. The communication consists of independent pairs of requests and responses. Such stateless protocols do not require the infrastructure to retain session information or status about each communications partner for the duration of multiple requests.

Stateless computing has certain implications in coding for the cloud. For example, if an outage occurs, instances of an application

could go away. Developers also need to understand that the internal state of an application is not going to be available as expansions are happening or as application components fail. Thus, application state should be stored external from the compute container in a database, object store, or message queue.

The goal is to make applications as stateless as possible. For instance, when communicating with services or subroutines within the application, developers should make sure the necessary data is passed to the application or enable the application to gather that information from the queue.

Most web applications do not store any information on front-end servers because, along with the back-end data layer, the front end usually needs to scale. To successfully implement stateless computing, developers need to change how they view their hardware needs, how they store and access information, and how they scale.

MAKE APPLICATIONS EVENT DRIVEN

An event-driven application is a computer program written to respond to actions generated by the user or system. For cloud-aware applications, an important strategy for designing for high availability is to code applications to read requests from a queue instead of synchronous calls. This allows multiple application instances to process requests. Instead of tightly integrated synchronous calls that each demand a precisely timed response, requests for work come off a queue, enabling adding multiple applications instances to process work that much faster. This can substantially improve application performance, depending on the additional instances added. What's more, if an instance is lost, the damage is minimal. The most that is lost is a single transaction. The remaining information in the queue continues to be distributed among the different active instances.

SCALE OUT, NOT UP

Scaling up always has a break point. If developers scale up, a point is reached where an application runs out of resources. Scaling up databases, for example, limits application performance to the database's server performance. If an application goes viral and attracts users in the millions, it will hit a break point. Scaling out (horizontally) ensures nearly infinite scalability. As long application bottlenecks are anticipated and dealt with in application design, applications designed to scale out can readily take advantage of the cloud's massive scalability and elasticity.

We scale database servers out through replicas of the primary database and through partitioning data to multiple database servers. Historically with traditional database servers, it was difficult to partition the data without extensive changes to applications. With current database technologies, we can implement shards that split data to their own database server with little or no changes to applications. This enables database servers to be partitioned and easily managed for backup, restore, migration, and maintenance. It is much easier to perform data replication or a backup on a 1 terabyte (TB) live database split into four different database servers of 250 gigabytes each than to experience the input/output performance degradation involved in backing up one large 1 TB database.

The complete separation of compute and persistence provides the best deployment and scaling flexibility. We prefer web services for all data storage, including log files and debugging streams. This shields the application from underlying changes in storage technology and decreases deployment overhead. With nothing stored locally on the compute instance that is running the cloud application, the application can run in an internal cloud, public cloud, or in both locations.

DESIGN FOR EVENTUAL CONSISTENCY

For data to be highly available, it must be replicated between sites and synchronized. However, it is important to balance this requirement against cost and performance. This is particularly true in eventual consistency, the condition in the cloud where a change in the database or application may not be registered for a few milliseconds. Since multiple application instances could live globally over multiple availability zones and include geographically distant zones, a cloud developer must consider how to handle inconsistencies from different instances drawing from the database.

One method is to shard data to manage scaling needs locally within the site and remotely to other sites. Each shard is a portion of a database consisting of one or more rows. Using what we call GEO shards, we write data for a specific geographic region to a specific designated site. Within each GEO shard, we create local shards to enable scaling within the site. To ensure high availability, we use replication to mirror each shard to another site.

For example, when an application is globally distributed using an active/active design pattern (described below), users accessing that application use an instance that is closest to them geographically. The data that is associated with that application is sharded across databases hosted at various sites. To enable this, a horizontal partition is created across database instances of the same table, resulting in multiple sites with different data. Eventually the data becomes consistent over time as updates are synchronized among the database instances.

In eventual consistency, an application needs to be able to handle latency in data with multiple sites. When the application is in the process of selecting data that is still in transit, it needs a retry method to accommodate the few milliseconds of latency.

Applications also need to manage uniqueness differently in eventual consistency. Since data is in multiple places, the application needs to be able to recognize uniqueness violations and adjust accordingly. If an application is trying to insert specific data that just got inserted in another site, the application needs to query the data to see if it exists in these other locations and adjust the update operation accordingly.

Eventual consistency isn't recommended for all applications, particularly those that require exact information at all times, such as an inventory system. For many applications though, eventual consistency works well. For example, the data involved in a global conference room scheduling application for a site in a particular geographic region would not often be accessed by people in a different region.

In some cases, developers will have to work with conflict resolution scenarios. Consider the conference room scheduling application example again. If someone traveling from one region to another location tries to reserve a conference room at the destination, a mechanism must be in place to handle the scenario where someone at the destination facility is simultaneously attempting to reserve the same conference room. Otherwise, both parties could potentially receive confirmation messages. This issue could be resolved with a time delay for confirmation or Paxos consensus protocols employing a number of message delays up to an agreed value.

ACTIVE/PASSIVE AND ACTIVE/ACTIVE DESIGN

We accommodate high availability needs and handle unpredictable demand two ways: using active/passive and active/active designs. In active/passive design, an application is deployed to two different locations or availability zones offering IaaS or PaaS (see Figure 2). Both these locations can run instances of the application, but only one location is active at a time. A local load balancer handles the localized application instances, and a global load balancer performs health checks to decide where to send the end-user requests. With the databases mirrored on the back-end, if one location is lost, the health check automatically fails over to the other location.

For global service availability, we prefer to use an active/active setup. In this case, both locations are active, running simultaneously, handling different users, and ready to fail over to each other should it become necessary. If one location fails, global load balancers help redistribute the load across the remaining locations. The advantage of active/active is improved utilization of cloud resources and a better ability to provide service availability closer to users through multiple active locations. Because each cloud is active, cloud applications must be written for active/active design, particularly with regard to handling conflict resolution.

Intel IT is already starting to work with active/active/active design patterns, which extend active/active concepts to three sites in a combination of public and private clouds.

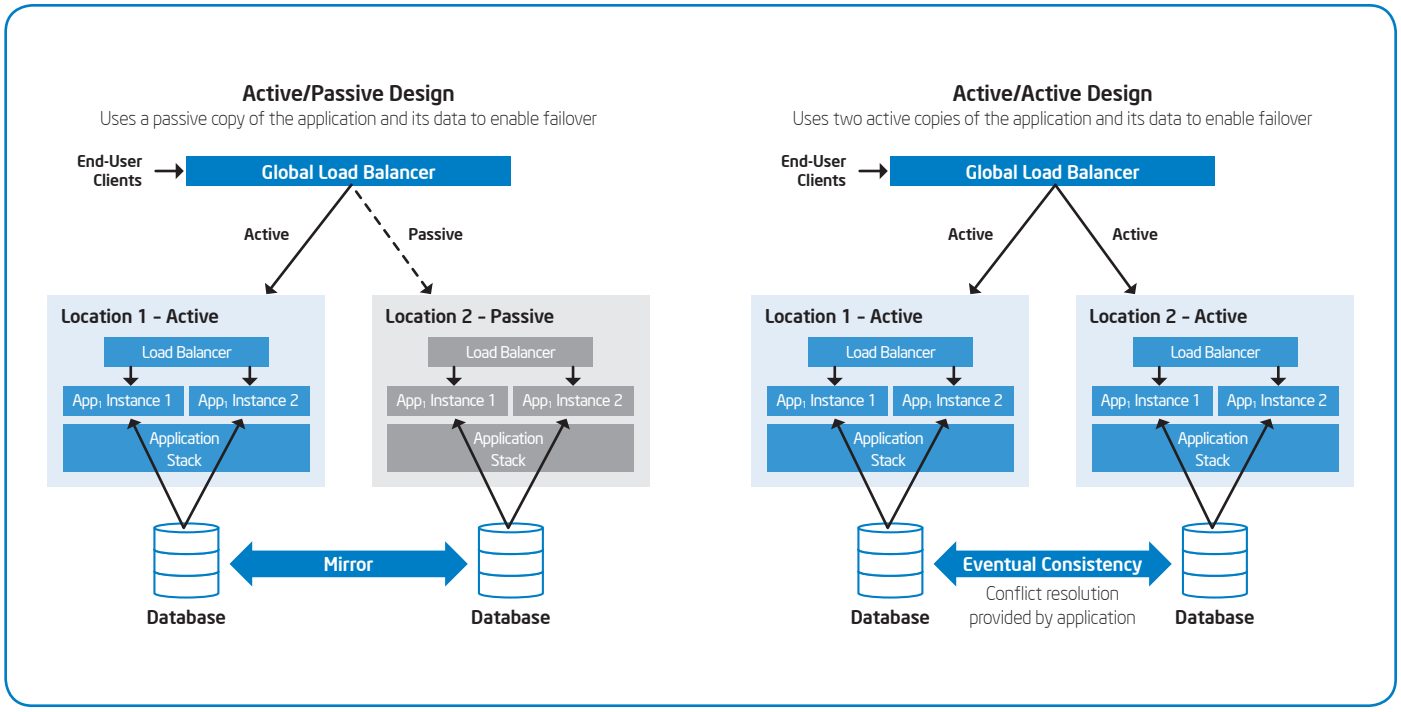

Figure 2. Our active/passive design, shown on the left, uses a passive copy of the database to enable failover. Our active/active design, shown on the right, uses two active copies of the application and its data to provide improved service availability, as well as enabling failover.

Security

When designing cloud-aware applications, we expect Intel developers to build security into the application, as well as to make use of security controls critical to the enterprise that are made available as consumable services. We do not assume that an application is secure because it is behind a firewall.

To help ensure the survival of the application on the Internet, developers and the business must require cloud service providers to follow best practices for providing a secure foundation upon which to build cloud-aware applications. We use security assurance methodologies that incorporate security design and testing processes.

Enterprise cloud environments should provide security services such as identity, authentication, and entitlement that focus heavily on integration and risk mitigation. Enterprise security capabilities developed as a service or through other methods allow an application owner to easily implement standardized controls to deliver the right balance of controls between the tenant application, the underlying cloud infrastructure, and the risk mitigation needs for the corporate business.

When designing for the cloud, we find that developers need to consider data handling and security within the application. Security should be layered and granular, using secure coding and access control inside the application, as well as authorization to determine if a user has the credentials to access specific information. We use encryption to ensure data protection in transit and at rest.

To ensure effectiveness, Intel application development teams routinely test and analyze the security of every API and data source their application will use or contact. Intel IT's security team is currently working on models for cloud that will enable us to host highly confidential, business critical, or regulated information in the cloud. Applications for these will need even more enhanced security and more exhaustive testing.

Teaching Cloud-Aware Development through Code-a-Thons

One successful way Intel IT is training developers is through hands-on training events. We host a series of Cloud-Aware Code-a-Thons where developers compete with each other to build the best cloud-aware applications and win prizes, such as a new Ultrabook™ device. Attendees develop cloud-aware applications, and then using platform as a service (PaaS), they quickly deploy applications in Intel's enterprise private cloud environment. We selected PaaS because it provides a shared, pre-provisioned environment that is designed to make it possible to land applications in less than a day.

Our one-day session format uses a variety of training elements.

- A presentation from the Intel IT cloud team on cloud-aware applications
- Hands-on coding of applications and practice landing them
- Roaming cloud experts providing "roadside assistance," including help in application development, security, hosting, and other important aspects of cloud-aware applications
- A judging session that chooses winners based on use of cloud-aware design principles—such as security in every layer and consumption of web services—as well as overall usefulness

These one-day code-a-thons produce interesting applications, such as the following:

- **Expert finder.** Links employees to Intel experts in particular areas. The application combines on-premise employee information with social computing profile information such as online resumes.
- **Cafeteria application.** Uses crowdsourcing techniques to provide up-to-date availability of certain items in an Intel café.
- **Parking applications.** Helps Intel employees find open parking spots at crowded Intel campuses.
- **Sabbatical countdown calendar.** Accesses employee information to determine the countdown to eligibility to each employee's next sabbatical.

RESULTS

Intel IT's progress in implementing cloud service models and providing architectural guidance and training for cloud-aware application development is changing the way Intel developers think and work. As a result, we are targeting more applications designed from the ground up to maximize cloud advantages and achieve faster time to market.

We have already reached a number of milestones.

- Identified over 400 IT employees that meet the criteria for needing cloud-aware development skills, helping us target communications to them and their managers.
- Established a training program specifically for cloud-aware application development as part of our "Skill up for the Cloud" career development materials. We are working on developing a way to measure proficiency.
- Delivered a series of brown bag sessions on specific topics related to cloud-aware applications, including Introduction to the Cloud, Cloud Hosting Options, and more.
- Hosted three code-a-thons in 2013, training 60 people so far in California, New Mexico, and Israel. Five more are planned for 2013. Trainees came from both Intel IT and various Intel business groups. More than 75 additional people have signed up for upcoming code-a-thons in Malaysia and India.

CONCLUSION

Intel IT is maximizing the value of our transition to cloud computing through an end-to-end model for cloud architecture. The continuing development and implementation of cloud-aware applications plays a critical role in unlocking this value.

Our adoption of learnings from grid computing and best cloud practices for consumable web services and high-availability designs is enabling Intel developers to quickly create applications from consumable services such as infrastructure, platform, and software. Equipped with a multi-platform front-end, these applications adapt seamlessly to a wide range of mobile devices.

In continuing our path to the Open Data Center Alliance's Cloud Maturity Model, Intel IT sees an ever-growing percentage of cloud-aware applications replacing legacy applications. IT organizations that want to take a similar path and maximize the value of their cloud efforts through cloud-aware applications can take several steps.

- Train architects and developers on cloud-aware application fundamentals.
- Create reusable application design patterns and templates.
- Promote the principles of cloud-aware applications. These include using web services to speed development, employing active/active design for high availability, and designing for failure to ensure resiliency.

RELATED INFORMATION

Visit www.intel.com/it to find white papers on related topics:

- "Accelerating Deployment of Cloud Services Using Open Source Software"
- "Benefits of a Client-aware Cloud"
- "Best Practices for Building an Enterprise Public Cloud"
- "Developing a Highly Available, Dynamic Hybrid Cloud Environment"
- "Extending Intel's Enterprise Private Cloud with Platform as a Service"

ACRONYMS

BYO	bring your own
CLI	command line interface
GUI	graphical user interface
IaaS	infrastructure as a service
JSON	JavaScript Object Notation
MTBF	mean time between failure
MTTR	mean time to recovery
PaaS	platform as a service
REST	representational state transfer
SaaS	software as a service
TB	terabyte
VM	virtual machine

For more information on Intel IT best practices, visit www.intel.com/it.