Design Patterns and Building Blocks

From HUIT Architecture Advisory Group

Contents

[hide]

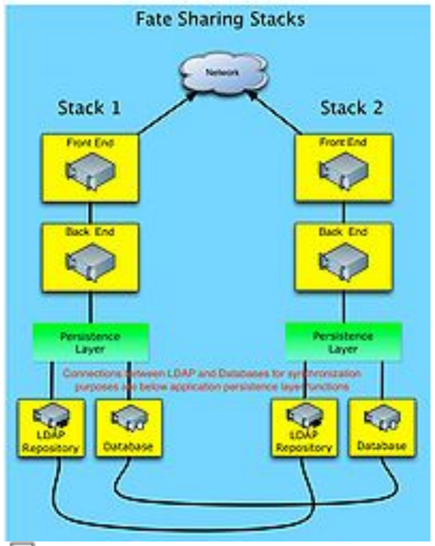
What are Architectural Design Patterns and Building Blocks?

Most of us are familiar with software design patterns. The same principle has also been used in enterprise architecture efforts. The NIH site cited in the Reference Information has a good description of design patterns in the context of an enterprise architecture. Both the software and enterprise architecture uses attempt to improve efficiency by using proven concepts and structures repeatedly. These design patterns must be realized with concrete technologies. These 'building blocks' are necessarily coordinated to work together in a design pattern. It is for this reason that we may have more than one design pattern with different building blocks to meet a general set of needs such as those in our three tiered highly reliable environment. In one case, the design pattern uses 'traditional' technologies found in most data center environments, while the other uses newer cloud technologies found in Amazon's AWS offering. See section 3.0 Collaboration Bricks page 18 of the NIH Enterprise Architecteure. While dated in some respects the section referenced shows what can drive the need to have different design patterns supporting a single use case. We have some systems and technologies that are not adaptable to new environments, yet we want to support them in the best way we can. In other cases, we are beginning new initiatives where we have the opportunity to impact the software architecure and technologies in a way that we can more cost-effectively provide reliable and predictable performance.

Design Patterns

The design patterns and building blocks referenced on this page are used to illustrate how the architecture can support specific business/technical requirements. The diagrams are often built to support identified use cases/scenarios. Also note that in some cases more than one design pattern could be used to meet the requirements of a particular use case. The Fate-sharing server stacks example could be used to meet the requirements of a future Learning Management Environment. Since that is a new effort, the High-Level Cloud Archiecture for Services with High Availability Requirements is probably the better choice representing the strategic direction of technologies within HUIT.

Using Fate-sharing Server Stacks - High Reliability and Scale with Traditional Data Center Technology

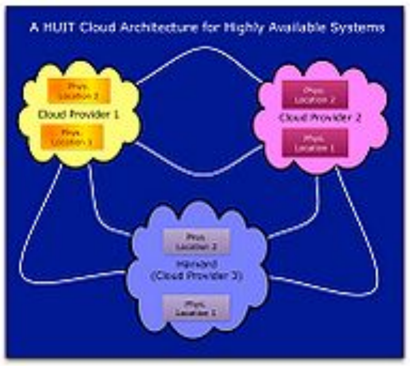


Fate Sharing Stacks

A basic concept and design pattern in the architecture is that of fate sharing stacks. In this fundamental pattern, each stack is independent of all the other stacks that support the service instance (for example a mail service). and that if one element of the stack fails, then the stack is out of service. As many stacks as are required for reliability and performance are deployed as are needed for the particular service.

Note that while this pattern can provide very good characteristics with regard to scalable performance and high reliability, additional faciliities for DR may be needed since it does not have the capabilities of an AWS environment where data can be synchronized in a geographic region unlikely to be impacted by a disaster in the primary region.

High-Level Cloud Architecture for Services with High Availability Requirements

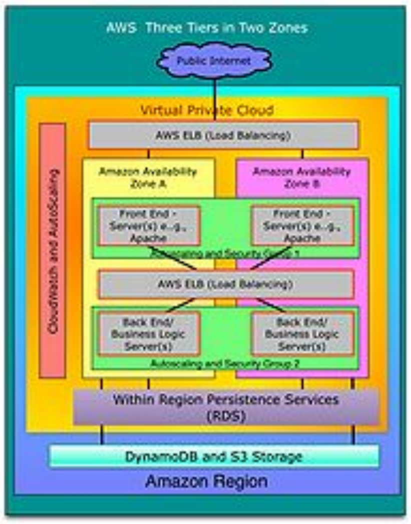


High-Level Cloud Architecture for Services with High Availability Requirements

This simple diagram illustrates how high availability applications might be deployed across different service providers. For these purposes, Harvard/HUIT is but one of several potential service providers. Note that each service provider may have multiple locations (and we may want to take advantage of them for certain mission critical services).

The diagram shows redundant paths between service providers. In some cases, a single service provider is all that will be required for some services and it may or may not be HUIT. In other cases there may not be connections between third party vendors for a variety of reasons. In this case, Harvard will connect to these other provides and provide required coordination.

Amazon - AWS Two Zoned Implementation for High Availability


AWS Three Tiers in Two Zones
Public Internet
Virtual Private Cloud
CloudWatch and AutoScaling
AWS ELB (Load Balancing)
Amazon Availability Zone A
Amazon Availability Zone B
Front End - Server(s) e..g., Apache
Front End - Server(s) e..g., Apache
Autoscaling and Security Group 1
AWS ELB (Load Balancing)
Back End/ Business Logic Server(s)
Back End/ Business Logic Server(s)
Autoscaling and Security Group 2
Within Region Persistence Services (RDS)
DynamoDB and S3 Storage
Amazon Region

AWS Two Zoned
Implementation

For some applications, high availability is important. This diagram illustrates the use of two Amazon availability zones within an Amazon Region. The availability zones are close enough to make practical synchronization of information across zones in different facilities. While two availability zones are illustrated, there are some regions with three zones as is the case on the east coast. In this event, the use of the third availability zone is not significantly more complex than two, though we do not know how meaningful that addition would be with regard to scalability or availability.

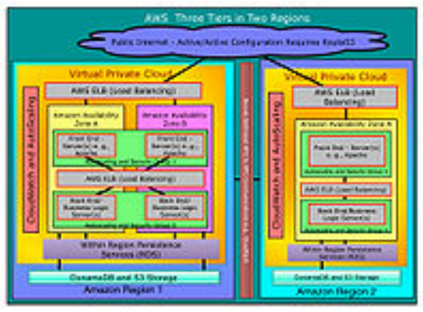
Note that in this example, there are red lines around the systems to indicate that they are protected. The method of protection is through the security group facility. The is easier to provision than individual instances of software firewalls on each host. Also note that there is a security group (though not drawn in the diagram) to which the external elastic load balancer is assigned. In this case it is a security group with a single member.

Autoscaling capabilities are across zones for a given tier of the application (e.,g the front end servers).

If there are applications that require close to continuous operations even in the event of a catastrophic failure of two availability zones within a region, they would require additional mechanisms not illustrated by this diagram.

Also note that there are a variety of storage technologies a service may use as is the case today. In some cases, a regular file system (which would be created with EBS in the Amazon environment) is all that is needed. Other cases may require a combination of standard file, relational DB, and perhaps a NoSql database for their persistence needs. The diagram illustrates that all variations would be available in specific implementations of this design pattern. Note that the S3 facility does work across availability zones, but is not contained within our VPC.

RDS and Multi-Region Operation


AWS Three Tiers in Two Regions
Virtual Private Cloud
Virtual Private Cloud
CloudWatch and AutoScaling
CloudWatch and AutoScaling
Amazon Region 1
Amazon Region 2

Two Region Operation

For a very few applications, there is a requirement for extremely high availability and a multi-region approach is desired. Consider the following in cases where such high levels are desired:

- The RDS layer includes both the code as well as the data for the database technologies supported in AWS. Applications make calls to this layer via APIs as they would on a physical system with a local database.
- RDS is a good way to get started with reasonable performance but there are certain

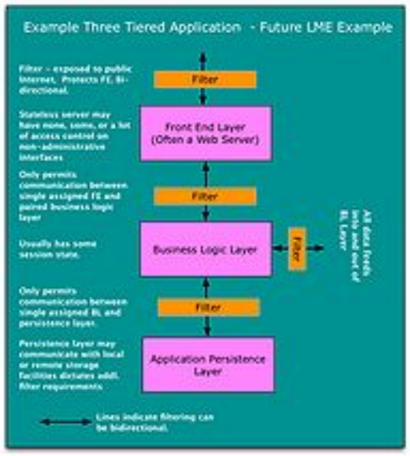
functions that are currently not permitted via RDS that may matter for many HUIT applications. For example, DB users are not able to be granted DB super user access. This would be needed (for example) to set up a master/master replication scheme. While this specific may change, for mission critical applications, we should assume a standard DB implementation in AWS that does not use RDS and as such would require DBA time to maintain and tune - just as it does today though our automation and orchestration approach evolution should produce efficiencies in the AWS and other environments. Cross region active/active databases are not feasable unless RDS is used - though this may change in the future as well. A standard DB implementation in this case means that the DB code is executing on one of our instances and is not a 'service' like RDS.

- While an active/active region configuration may be desired in a very few applications, a simpler approach might be to have an active/standby approach as described in the next point.
- Create 3 or 4 availability zones in a region with 1 'stack' in standby mode in a second region that would contain a script to promote that region to active in the event of failure. These should be 'reserved' instances to increase the likelihood of our getting the resources we need in the unlikely event of an entire regional failure. (as an aside, I like this model for a few of our other most critical applications).

VPC to VPC Communications

- The best way to achieve VPC communications (inter and intra region) is via solutions like OpenVPN or Sophos which is also and EC2 AMI.
- In those cases where we had real time feeds for database read replicas, etc. we could use CloudOpt.
- AWS is working on a VPC to VPC communications product that would simplify this for us and eliminate the need for these specialized products.

Three Tiered Application with Access Filters


Example Three Tiered Application - Future LME Example
Filter - exposed to public Internet. Protects FE. Bi-directional.
Stateless server may have none, some, or a lot of access control on non-administrative interfaces
Only permits communication between single assigned FE and paired business logic layer
Usually has some session state.
Only permits communication between single assigned BL and persistence layer.
Persistence layer may communicate with local or remote storage facilities dictates addl. filter requirements.
Lines indicate filtering can be bidirectional.
Filter
Front End Layer (Often a Web Server)
Filter
Business Logic Layer
Filter
All data feeds into and out of BL Layer
Filter
Application Persistence Layer

Three Tiered Application
with Access Filters

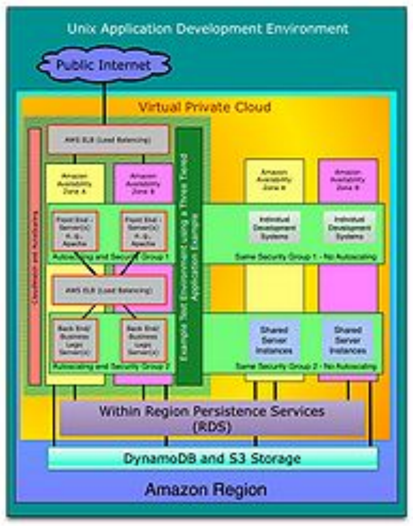
This diagram illustrates a typical three tiered application and the placement of filters around all the main components. Note that the front end will generally be reachable by anyone and is in the public address space. While the front end will often be a stateless Web server, other front ends with this design pattern are possible. They include: load balancers, SSL termination devices, NAT facilities, authentication gateways or even a Microsoft Exchange (CAS) front end. What is important here are the filter rules and the fact that no caching of level 3/4 data is permitted in these front end systems. This pattern is suitable for systems that have Level 1, 2, and 3 data. The only difference for a system that would contain Level 4 data is that interfaces on systems with Level 4 data must not be in the public address space. Please also note the following:

1. Only the business logic layer may access its application persistence layer. This means all feeds into and out of the 'stack' are through this layer, not the persistence layer.
2. For those applications that require multiple 'stacks' for reliability purposes, functions that sync data are handled by lower level services such as: LDAP Duplication, Database duplication or Cloud/Vendor specific technologies.
3. Some technologies/vendors have approaches that provide the same protection but use a different approach. In the case of AWS, they would not use these 'filters' which may be implemeted as host level firewalls. Instead, they would use security zones control which systems have access to other systems. This is a layer around the VM. For an example using AWS technologies, see Amazon - AWS Two Zoned Implementation for High Availability

---

Unix Application Development Environment with Example Three Tier

Application

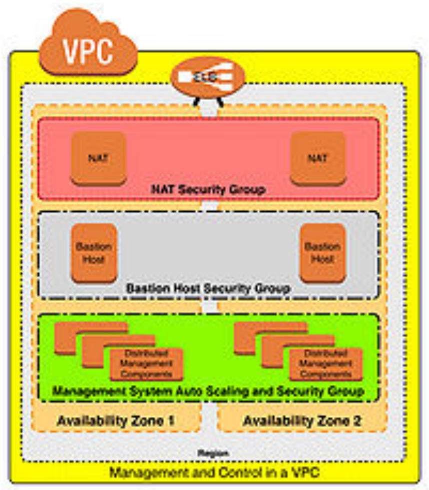

Unix Application Development Environment
Public Internet
Virtual Private Cloud
AWS ELB (Load Balancing)
Amazon Availability Zone A
Amazon Availability Zone B
Front End - Server(s) e.g., Apache
Autoscaling and Security Group 1
AWS ELB (Load Balancing)
Back End/ Business Logic Server(s)
Autoscaling and Security Group 2
Example Test Environment using a Three Tiered Application Example
Individual Development Systems
Same Security Group 1 - No Autoscaling
Shared Server Instances
Same Security Group 2 - No Autoscaling
Within Region Persistence Services (RDS)
DynamoDB and S3 Storage
Amazon Region

Unix Application
Development and Test

This diagram reuses the three-tiered application production environment design pattern. In this example, a project that is developing Unix application software exists in the cloud. The plan is to deliver the application(s) in the cloud as well. The left side of the diagram represents a full test environment where as many instances of test systems as are necessary to support the engineering and test effort may be spun up as needed. On the right side are the individual engineering development and basic test systems along with supporting systems like, build systems, specialized logging or test systems, etc.

This is a good example of how a general design pattern could be modified/created through the full or partial inclusion of other design patterns already established to save significant amounts of time and resources. Even this case could be further modified from this known starting place if needed, for example different projects may have different specialized tools requirements.

AWS Management and Control Plane


VPC
ELB
NAT
NAT
NAT Security Group
Bastion Host
Bastion Host
Bastion Host Security Group
Distributed Management Components
Distributed Management Components
Management System Auto Scaling and Security Group
Availability Zone 1
Availability Zone 2
Region
Management and Control in a VPC

Management and Control
in a VPC

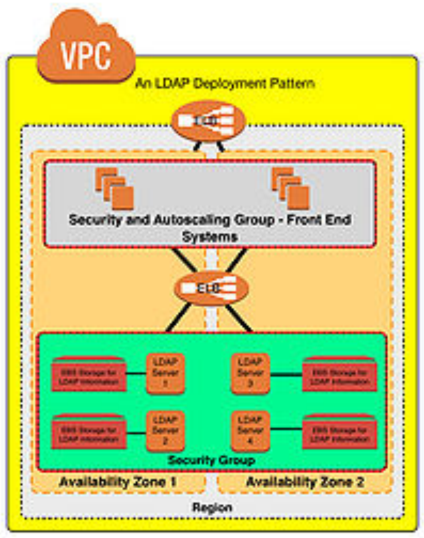
Separation of management and control from the main data path have been best practice for some time. This design pattern adapts this best practice to the specifics of AWS, but it could be translated to a number of different environments.

The diagram illustrates separation of functions within management and control. The NAT functions are separate from the Bastion hosts. This approach also helps us limit each system type to their essential functions. Note that separate security groups for the NAT and Bastion hosts have been used since they will most likely have meaningful differences with regard to external access permissions. The Management Systems can be placed in an autoscale group (whereas this would be problematic and is not needed for either the NATs or Bastion hosts).

In the case of a deployment across regions, this model still applies. Each AZ in the second region would look like the AZ in the single region above.

This article describes how to create a high availability environment for NAT systems.

High Availability Deployment of LDAP in AWS


VPC
An LDAP Deployment Pattern
ELB
Security and Autoscaling Group - Front End Systems
ELB
EBS Storage for LDAP Information
LDAP Server 1
LDAP Server 3
EBS Storage for LDAP Information
EBS Storage for LDAP Information
LDAP Server 2
LDAP Server 4
EBS Storage for LDAP Information
Security Group
Availability Zone 1
Availability Zone 2
Region

High Availability Deployment of LDAP in AWS

This pattern illustrates how a multi-master (4 in the case of this diagram) LDAP deployment can be achieved in AWS. The LDAP servers are not in an autoscale group because of the configuration requirements of each LDAP Master instance. Also note that the LDAP information is in persistent EBS storage associated with each LDAP Master instance. These EBS instances can be reattached to other instances within the same availability zone when those instances start up in case there is a failure of one of our LDAP server instances. This pattern shows the servers behind an ELB and as such has the assumption that port numbers can be remaped. The reason for this is that at the time of this writing, AWS ELBs had restrictions on the ports/protocols that they could work on. Currently the ELB supports ports 25, 80, 443 and 1024-65535. The restriction is similar to load balancing products from other vendors. In those instances where direct external access to LDAP is needed (assuming the security groups and NAT system were configured to meet security requiremnts of the data in the repository) a separate LDAP instance could be configured for this access. In addition the Route 53 product could be used to do remapping with the fail over facility. For example, if the request to the desired DNS name/port combination failed, it can remap to an alternate. This would work across regions which is a nice feature.

Other Design Patterns

People that have been working with Amazon Web Services have created a site that has a variety of design patterns that could be used to augment those developed here. Additional ideas for topics not covered on here may be found at that site.

Retrieved from "https://wikis.fas.harvard.edu/huitarch/Design_Patterns_and_Building_Blocks"