Use Cases/Scenarios

From HUIT Architecture Advisory Group

Contents

[hide]

HUIT Use Cases/Scenarios

A key strategy used to determine the enterprise architecture is to identify a number of use cases that cover a broad range services. The use cases on this page are often based on existing services provided by HUIT, though many current implementations would require significant work to operate in the new environment. Whether refactoring or complete replacement is appropriate is a separate analysis that would have to be undertaken on a case by case basis.

Use cases this page are the combination of the current cabapilities refactored into a new architecture made with common building block and design patters. These uses cases are also based on the assumptions and rules and procedures from an architectural perspective that would govern their operation along with enhancements possible through the use of newer operational and design approaches.

Use Case Development Approach

For each of the use cases we started with services such as iSites, email, or logging services. Please see the Glossary for a discussion on the distinction between infrastructure services and customer visible services. The services described from these use cases, represent each type of service and in the case of email, it can act as a customer visible service or an infrastructure service. From this general view we examined the service from a number of perspectives to more fully understand the requirements of the system. Since each service is different, not each of the factors below will receive the same weight or even be considered. In other cases, special considerations drive the need for describing additional perspectives relevant to that service. In general, the items below were considered for each use case to determine relevance and whether additional factors were needed:

1. Security - What types of data did the system process and what types of data did the software store? This includes the feeds into and out of the system.
2. Availability - How critical is the application? This could range from, can go down infrequently with no real impact to it must never go down. It could be expressed as how many nines are we willing to pay for to keep this application available. This is distinct from issues related to data synchronization under storage requirements in this list.
3. Scale - in this case means issues related to number of concurrent sessions, computational power required, and what useful increments would be. This is disctinct from issues related to storage scale and the specific requirements found there.
4. Performance characteristics - closely related to scale and user iteractions, this set of requirements related to specific measures that indicate user customer expectations of the system overall. For example, a response screen in N seconds after a commit or a screen refresh in Y ms.
5. Feeds and other dependencies - what are the characteristics of each of the feeds:

    1. Source
    2. Destination/Consumer in the case of systems that depend on feeds from the system
    3. Frequency - how often are they updated or a continuous stream
    4. Volume of data
    5. Failure conditions - in the case of a failed feed or issues consumers may have in the case this applicatoin fails
    6. Feeds on which the service depends
    7. Applications or services that depend on data in this system
    8. Authentication mechanisms
    9. Other infrastructure services
6. User Interactions - this section is intended to capture requriements related to different client types, user scenarios with the application, etc.
7. Storage Requirements
    1. Scale - Baseline requirements, expected growth
    2. Types - e.g., basic file, LDAP, DB, video, other
    3. Read and Write Characteristics
    4. Data Synchronization
    5. Data Synchronization, Timeliness
8. Archival and Data Retention
9. Requirements related to disaster - if there is a total loss at one location, how soon before services are restored to full operation? Is there a requirement for never being down (e.g, different availability zones, or different availability zones in different regions?

Use Cases

Taken together, the use cases in this section are believed to represent a sufficinet diversity of requirements so that if we can create design patters for each, we should be able to address existing services not covered by them with relatively simple adjustments. As new services are created, one or a combination of the design patterns based on these use cases should be abble to address the new services.

Generic 3-tier Application

We used the current iSites system along with anticipated requirements for a new generation learning management environment as the source of requirements for this use case. It is also informed by an understanding of new technologies that may be helpful in constructing a system to meet the availability, scale and other requirements described below. The AWS Two Zoned Implementation is intended to represent the design pattern that would meet the requirements of

this type of application.

Availability from the user point of view

The system has a large and diverse population and is always in use. There are periods of peak demand such as the beginning and end of terms. It is less critical at other times such as the day after commencement. It is also the case that these peaks and troughs are not universal. Even in very low use times, a few users may have need of the system. An example given was a upgrade the weekend after commencement. This upgrade put the system in read only mode. The law school was conducting course and people had flown into country for the law school course. The read only status of the system at this time was a problem for the course in session. MOOCS always need to be available as is the extension school.

For some end-users there is no value in the distinction between scheduled and unscheduled. That said, a goal is to minimize planned outages - for example server upgrades. There are some conditions that the architecture and software design may not fully anticipate that would require a planned outage or reduction in service, but these should be rare edge condition rather than the general practice.

Short term outages of up to 120 seconds are OK, this will have minimal impact as long as state is not lost, and as long as those short term outage are not very frequent.

To avoid outages as a result of failover of components, or certain upgrade conditions, the follow specific requirements apply:

- Want to share state across systems (session).
- Want a persistence layer that can be upgraded.
- Want to do upgrades without significant outage or having to do them office hours.

Scale

Must scale from a few tens of users to 100,000s of active users per day. Currently there are 1,600 users at any one moment with a peak of 5000. Most of the activity is read and there is light write activity. Most data are stored in a database. Files are uploaded into individual file systems. There are many 10.000s individual transactions per minutes. The vast majority of these are read transactions. In order to give the system flexibility, a variety of persistence layer technologies should be available in

addition to the other approaches to building scalable systems such as Fate Sharing Server Stacks.

Performance from a User Perspective/Response Time

If a session goes away the data should not be lost. If, as a security measure, the session is timed out, the data should not be lost either. When the user re-logs in the data should be recovered. Average response times of 2 - 4 seconds is OK from the user perspective.

From a user perspective performance/response time consistency across different load levels is hard to achieve but should be a goal.

Feeds

This system will have feeds from/to a number of sources, Key among them are:

- From the registrars offices at each of the Schools.
- Feed from IDDB.
- Different IT departments e.g., HR or Library.

As a rule, the system will use stale data in the event of an update failure. At present feeds are generally periodic such as each 24 hours. The system should allow for the possibility of real time feeds where appropriate (see next section).

Infrastructure Services

Most infrastructure services have real time requirements. Examples on which this use case depends include:

- Authentication - at present PIN is used, later it may be CAS, Shibboleth. Active Directory, LDAP or eCommons.
- Attribute service provides you something about the individual (what is the ID for the person).
- Authorization function - One example is the current AuthZProxy, but others will be used in the future.
- Email-address comes from IDDB or LDAP which could be any email address anywhere. For sending email, use the SMTP service from the unix group. There can be no assumptions about which mail system environment at the University

the individual belongs. Some participants may not be DCE members or have non-Harvard email addresses. I may also be the case that some applications may want to send email directly from their environment. One example would be the AWS Simple Email Service.

User Interactions

While people can upload documents today, a more interactive mechanism is desired, People would like the facility to drag and drop items from the 'folder' on their local device to the course site. Conversely, they would like to be able to download elements from the course site as well via a drag an drop approach.

Storage Requirements

1. Low performance file storage where it is infrequently accessed but must always be on line and in real time. If any one of these files were missing there would not be a significant consequence (except to potentially a few users). These files will be streamed over the web. There are a few TB of this type of data.
2. A smaller set than the few TB noted above, but critical to the operation of the system, e.g. .css files. Without these files, the system does not work.
3. Database very high i/o needs to support overall 2 second response time. If slow or unavailable thousands of users would be impacted.
4. Storage streamed for video some might need to be cached. This will become critical to the mission. Currently there are about 1000 courses that use video and about 20 students per course. About 20K users. We need to think in terms of PBytes of on-line storage.
5. Types - from the above it is clear there are a wide variety including: basic file, LDAP, DB, and video.
6. Archival and Data Retention - Need to be able restore from non-system connected storage to full operation. The replication of the data may be local. The issue here is to protect against an inadvertent or malicious attempt to delete key data.
7. Requirements related to disaster - There is an important requirement for the LME to be geographically diverse this is especially true in the mooc concept.
8. Data synchronization issues - The writable persistent data store is not written all that much. If you were to aggregate all the data in a day that is written it currently might be 10 - 50 Gb per day - but they need to reasonably current and in the event of a discontinuity in the system for any reason, we do not want to

loose any transactions. This does not include the video data. For the video data, a lag of up to a day for syncrhonization might be OK. This could be true for the low critiacly data. Lags do not mean loss. A special case of data includes key administrative transactions that are relatively smaller in number but which need to be synchronized withing seconds. Audit and logs can have a high latency. Would like the fail back to the main site (this might be region in AWS speak, or service provider) to be as fast and easy as the the fail over. Should be prepared to minimize a interruptions in both directions.

Backup Service and DR Use Case

Including client server.

Infrastructure Service

Every one of our 'business level' services like iSites, the payrol system and email, depend on many infrastructure services. Because of the unique requirements of extending across services, they present unique challenges. For this reason, we will use SPLUNK, our logging facility as an example infrastructure service. It is also valuable since it collects a lot of data, the majority is write once and read many.

Authentication Service

CAS

Large/ERP Applications

Peoplesoft, Oracle Financials

Non-Confidential Information General Purpose Application

Collaboration site is my example

Complex Storage Application

Reference HUIT Unix Application Development Environment

In order to support HUIT's key objectives and deliverables, its software engineering

environments should foster efficient communications between what may be transitory groups of engineers while enabling an environment that is flexible and yet relatively low overhead for each engineering team. There are some important differences between this environment and other environments HUIT services support. Chief among them is flexibility. While there are some elements of software/systems engineering that are common and should be followed regardless of the project (such as documentation, unit and systems testing, and requirements collection), team composition, objectives, technologies, the need to learn and adapt to new technologies, and the desire for innovation, require a bit different approach than some of our other services.

The requirements described in this use case are for individual engineers working alone or in combination with others on a single project. Left out of this use case are HUIT-wide systems.

Availability

While failures in these environments are not as visible as a failure of one of our major systems that may impact thousands of faculty, staff and students, lost productivity should be avoided. A single region with two availability zones and a real real-time feed to another U.S. region is sufficient. The reason for two availability zones is not the same as for our production systems. It is to provide an environment for testing that is closer to the production environment for full systems testing. Many of our past production failurs are the result of failures found only when deployed for production. The second region need not be pre-provisioned and ready to go, it the data for reconstructing the environment(s) is all that is needed.

Scale

Scale characteristics for this environment are a bit different than when considering a production system. In this case, it is the number of servers that will need to scale even when individual systems are not fully loaded. The objective here is to provide engineers with dedicated systems for testing (in many cases entire stacks may be needed). The nature of many of our systems is that the work to configure a system to allow multiple instances of the same service to run on it outweighs what it would cost to spin up another VM or two. The rate at which these systems come and go will be driven by the rhythms of the project (e.g., a time of system testing), new employees and projects, and not for example, number of transactions.

Flexibility

This environment is characterized as primarily self-service. That is, within the budgetary constraints set up by engineering management, individual engineers should be able to spin up additional instances of systems for development or stacks as necessary for testing purposes. Stacks in these environments may be customized to reflect particular configuration requirements of the application including requirements for different types of persistence.

Users should have sudo access on all of their machines (including test systems) under their control.

Security

Systems used for software development and testing must not contain any information above level 2. There should be no password, or key information stored anywhere in this environment. These systems must still be protected via software firewall and each image will contain a default configuration that will be adjusted as necessary.

It is also important for developers to be able to have remote desktop functions on the development and test systems.

Performance

There will be a basic system designed to give good performance for software engineering. The amount of memory will be defaulted to 16GB for engineering. Test systems will need to be tunable so that testing in a variety of conditions is possible (and may help simulate load conditions). Actual deployed size will vary based on individual project needs.

Tools

The basic reference implementation should have all of the basic elements an engineer would need. In this case, it may be simpler to have a couple more tools than a single engineer may use. For example, not all engineers would necessarily develop database software, but they would need it for testing purposes. An initial set for this reference implementation is:

1. Current Red Hat compatible OS with all current updates
2. Git
3. JBOSS

4. Standard editors - Vi, VIM, Emacs
5. Integrated Development Environment - in this case, Eclipse and Egit and Maven Plugins
6. Jenkins
7. Up to date javac/JRE
8. MySql
9. System configuration encapsulation as code via Puppet facility.
10. Tools for controlling the virtual environment -- for AWS CLI tools for managing the infrastructure.
11. Software Firewall with default configuration
12. Intra and Inter project collaboration tools like a Wiki and messaging facilities.

This use case is an example. Different departments will have variations in one or many of tools.

Storage

The amount of disk space will vary based on project requirements for this use case it is M1.medium. It is assumed that all meaningful data in the project is stored in Git which can serve both as backup and a way to ensure continued operations in a disaster.