in real life

Architectures for Cloud Applications

Mikael Svahnberg[1]

[1] Mikael.Svahnberg@bth.se
School of Computing
Blekinge Institute of Technology

April 30, 2015

Outline

- Recap
  - Software Architectures
  - Cloud Relevant Quality Attributes
- Architecture Implications
  - Scalability
  - Reliability/Availability
  - Performance
  - Security
  - Privacy
  - Cost Optimisation
  - Maintainability / Developability
- Summary

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

Software Architecture Defined

- First Definition: Boxes and Lines
  - What is the nature of the elements (boxes)?
  - What are the responsibilities of the elements?
  - What is the siginificance of the connections (lines)?
  - What is the significance of the layout?
- Second Definition: Add semantics (provide legend)
  - What is the significance of the layout?
  - What are the interfaces of the elements?
  - How does the architecture operate at runtime?
  - How do we build it?
- Third Definition:

  > *The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.*

Architecture Decisions

- Balancing all stakeholders result in a number of *Business and Technical Decisions*
- Software architecting is about identifying which decisions are necessary, and finding solutions that satisfy all stakeholders.

Decisions *are* the Architecture

I would go as far as to say that these decisions *are* the architecture. . . . The rest is just an instantiation of the architecture.

Influences on Architecture

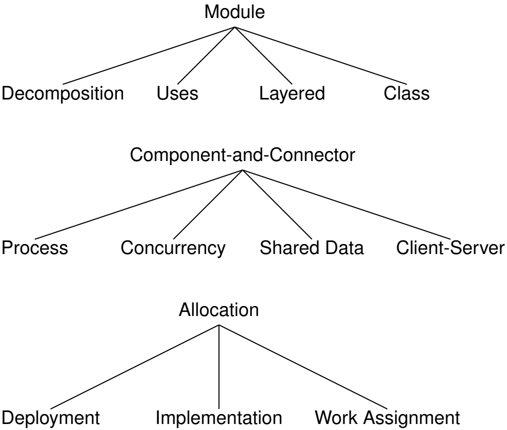
- Customer Requirements, of course
- Developing Organisation
  - e.g., business goals
  - Organisational structure
  - Available expertise (the architect’s experience)
- Technical Environment

A “Good” Software Architecture

- Is based on *conscious* decisions
- Is *evaluated* to ensure that it satisfies the specific goals for the system
- Pays attention to current and future *quality attributes*
- Is well *documented*, with traceability to the architecture decisions
- Features well defined *modules*(components), with well defined *interfaces* and well defined *responsibilities*
- Is restricted to a small set of interaction patterns that are *consistently* used.

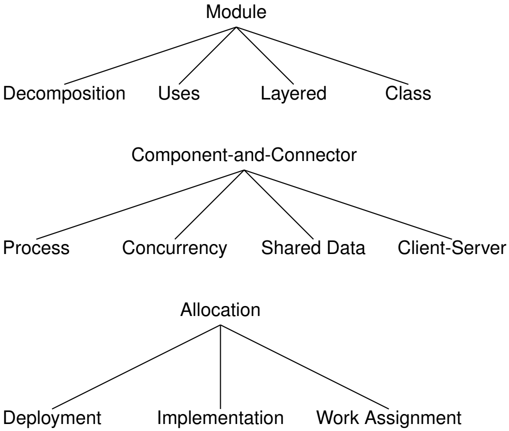
Structures and Views

Bass et al.(2012):



Structures and Views

Bass et al.(2012):



Module

Decomposition

Uses

Layered

Class

Component-and-Connector

Process

Concurrency

Shared Data

Client-Server

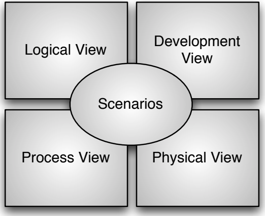
Allocation

Deployment

Implementation

Work Assignment

Kruchten(1994):



Logical View

Development View

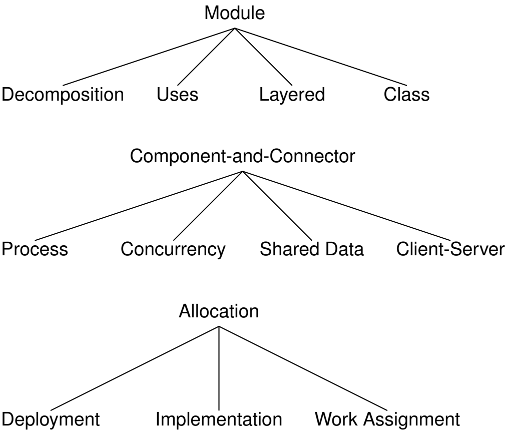
Scenarios

Process View

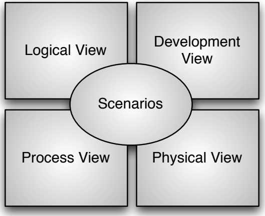
Physical View

Structures and Views

Bass et al.(2012):



Kruchten(1994):



Hofmeister et al.(2000) uses a variant of this:

- Conceptual View
- Module View
- Execution View
- Code View

Architecture and Quality Attributes

- Functionality is “easy” to implement.
- Quality requirements may *sometimes* have impact on the implementation
- More often, it impacts the *software structure* (=the software architecture).
- . . . And yet, the architecture can only describe a *potential* for achieving a particular quality level.

Outline

Recap

- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications

- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Factors that “push” you towards the cloud

- Transference – Move your on-site solution as-is to the cloud for e.g. economic reasons.
  - Challenges: Setting up a similar environment in the cloud as you have locally.
- Internet Scale – Scaling up to handle more users.
  - Challenges: Database design may become a bottleneck.
- Burst Compute – Large swings in capacity requirements.
  - Challenges: Strategy for load balancing, database access.
- Elastic Storage – Scaling up to handle (much) more data.
  - Challenges: need also to consider where the data is processed.

Cloud Relevant Quality Attributes

- Scalability
- Reliability / Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Architecture Implications

- This is a boring part of the lecture.
- Basically, we go through each of the cloud relevant quality attributes and discuss the corresponding tactics.

Important

The most important thing for you to think about, and for us to discuss is:

How would I address these issues and tactics in my Software Architecture?

Outline

Recap

- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications

- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Scalability

- Horizontal Scaling – add more nodes
- Vertical Scaling – increase capacity of nodes
- In a Cloud system, you would do both.
    - It is easier to scale vertically unless you already have a horizontally scaled solution
    - Storage space is often easier to deal with by scaling verically – unless you already have a horizontally scaleable solution in place.

Scalability Tactics

Bass et al. does suprisingly not have any tactics associated with Scalability. They list the following tactics for *Performance*:

- Control Resource Demand
- Manage Resources

Scalability Tactics

Bass et al. does suprisingly not have any tactics associated with Scalability. They list the following tactics for *Performance*:

- Control Resource Demand
  - Manage Sampling Rate
  - Limit Event Response
  - Prioritise Events
  - Reduce Overhead
  - Bound Execution Times
  - Increase Resource Efficiency
- Manage Resources

Scalability Tactics

Bass et al. does suprisingly not have any tactics associated with Scalability. They list the following tactics for *Performance*:

- Control Resource Demand
- Manage Resources
  - Increase Resources
  - Introduce Concurrency
  - Maintain Multiple Copies of Computation
  - Maintain Duplicate Copies of Data
  - Bound Queue Sizes
  - Schedule Resources

Scalability Tactics

Bass et al. does suprisingly not have any tactics associated with Scalability. They list the following tactics for *Performance*:

- Control Resource Demand
- Manage Resources

Discussion

What else can be done to support scalability *in the application*?

in real life

Outline

Recap

- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications

- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Reliability/Availability

- Primary tools: Redundancy, Geographical (and provider) distribution, load balancing.
- Cloud solutions allows for an informed trade-off between programming reliability in, and throwing redundant servers at the problem.
- Some Cloud Providers' only availability promise is that *your node will fail after some unspecified time*!
- Connected to data persistence, since you cannot expect a “responsible” node shutdown.

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
- Recover from Faults
- Prevent Faults

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
  - Ping/Echo
  - Monitor
  - Heartbeat
  - Timestamp
  - Sanity Checking
  - Condition Monitoring
  - Voting
  - Exception Detection
  - Self-Test
- Recover from Faults
- Prevent Faults

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
- Recover from Faults
    - Preparation and Repair
    - Reintroduction
- Prevent Faults

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
- Recover from Faults
    - Preparation and Repair
        - Active Redundancy
        - Passive Redundancy
        - Spare
        - Exception Handling
        - Rollback
        - Software Upgrade
        - Retry
        - Ignore Faulty Behaviour
        - Degradation
        - Reconfiguration
    - Reintroduction
- Prevent Faults

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
- Recover from Faults
    - Preparation and Repair
    - Reintroduction
        - Shadow
        - State Resynchronisation
        - Escalating Restart
        - Non-Stop Forwarding
- Prevent Faults

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
- Recover from Faults
- Prevent Faults
    - Removal from Service
    - Transactions
    - Predictive Model
    - Exception Prevention
    - Increase Competence Set

Availability Tactics

Bass et al. list the following Availability Tactics:

- Detect Faults
- Recover from Faults
- Prevent Faults

Discussion

Which of these would be particularly relevant in a cloud application? Why?

Outline

Recap
- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications
- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Performance

- The way you tackle performance depends very much on what type of performance you are after.
- For example,
  - Response time is very different from
  - Processing time, which is very different from
  - Storage capacity
- [In many applications] you probably have a mixture of many different requirements.

Example: Web application

- Response time: System must feel “snappy”
- Processing time: Behind the scenes, you may have activities that takes several seconds to perform.
- For example, submitting a post in a discussion forum may include:
  - Re-baking the user’s profile
  - Looking for cross-posts and re-baking these posts
  - Re-generate a thread summary
  - . . .
- In a sufficiently frequented forum, each of these actions may take several seconds to perform.

Example: Cloudbursting

- At the other end of the spectrum you have batch-processing applications
- You use the cloud's computing resouces to (re-) generate massive amounts of data.
- Response time is not an issue

Performance Tactics

- Control Resource Demand
- Manage Resources

Performance Tactics

- Control Resource Demand
  - Manage Sampling Rate
  - Limit Event Response
  - Prioritize events
  - Reduce Overhead
  - Bound Execution Times
  - Increase Resource Efficiency
- Manage Resources

in real life

Performance Tactics

- Control Resource Demand
- Manage Resources
    - Increase Resources
    - Introduce Concurrency
    - Maintain Multiple Copies of Computations
    - Maintain Multiple Copies of Data
    - Bound Queue Sizes
    - Schedule Resources

Performance Tactics

- Control Resource Demand
- Manage Resources

Discussion

Which of these would be particularly relevant in a cloud application? Why?

Outline

Recap
- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications
- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Security

- Security from External threats
- Security from threats inside the cloud provider
  - Covert channels between your VM and others'
  - Sniffing network communication
  - Inherently unsafe designs (e.g. Amazon S3's global namespace for their buckets)
  - Legal issues

Security Tactics

Bass et al. list the following Security Tactics:

- Detect Attacks
- Resist Attacks
- React to Attacks
- Recover from Attacks

Security Tactics

Bass et al. list the following Security Tactics:

- Detect Attacks
    - Detect Intrusion
    - Detect Service Denial
    - Verify Message Integrity
    - Detect Message Delay
- Resist Attacks
- React to Attacks
- Recover from Attacks

Security Tactics

Bass et al. list the following Security Tactics:

- Detect Attacks
- Resist Attacks
  - Identify Actors
  - Authenticate Actors
  - Authorise Actors
  - Limit Access
  - Limit Exposure
  - Encrypt Data
  - Separate Entities
  - Change Default Settings
- React to Attacks
- Recover from Attacks

Security Tactics

Bass et al. list the following Security Tactics:

- Detect Attacks
- Resist Attacks
- React to Attacks
  - Revoke Access
  - Lock Computer
  - Inform Actors
- Recover from Attacks

Security Tactics

Bass et al. list the following Security Tactics:

- Detect Attacks
- Resist Attacks
- React to Attacks
- Recover from Attacks
    - Maintain Audit Trail
    - Restore
    - (See Availability)

Security Tactics

Bass et al. list the following Security Tactics:

- Detect Attacks
- Resist Attacks
- React to Attacks
- Recover from Attacks

Discussion

- Which tactics can you automate? How?
- This covers external threats. What about the threats from inside the cloud provider?

Outline

Recap
- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications
- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Privacy

- *Extremely* important
- The easiest way to get bad publicity is to neglect privacy
- Often regulated by law.
- Local laws, that may differ between countries.
- Ties in with Security: Low security makes it harder to enforce privacy

Privacy Tactics

Not covered by Bass et al. Generic guidelines include:

- Restrict Stored Personal Information
  - What information do you need to store about your users?
  - Why?
  - For how long?
- Encrypt Data
- Use Secure Connections
- Hire a lawyer!
  - Strange as this may seem, this is a cost optimisation. If you do not have to have a replica of your system in each country you cater for just to satisfy local laws, the cost of the lawyer will be recovered quite quickly.

Outline

Recap

- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications

- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Cost Optmisation

- The whole purpose of a cloud solution is to optimise CAPEX vs OPEX.
- The easy solution is to throw more resources at the problem.
- For obvious reasons, this is not a sustainable solution long-term.
- *However*, there is a trade-off between how much time your developers should spend on optimising your application and the cost of adding an extra, or a larger cloud resource.

Outline

Recap
- Software Architectures
- Cloud Relevant Quality Attributes

Architecture Implications
- Scalability
- Reliability/Availability
- Performance
- Security
- Privacy
- Cost Optimisation
- Maintainability / Developability

Summary

Maintainability / Developability

- Not a cloud issue *per se*, but becomes more obvious when you are paying for uptime.
- Each of your developers need a development platform (as usual)
- Test Environment
  - How many test platforms do you need to support? One per developer? One per team?
  - How do you provide this? Local Virtual Boxes? On the Cloud?
- Staging Environment
  - As close to your deployment platform as possible
  - For extensive “release-testing”
  - How many do you need? Is one enough?
  - How do you test the system in the staging environment? Do you need to have a test harness environment too?
- Deployment Environment
  - Do you have just one deployment environment? Or is it one per customer?
  - How do you elastically scale this environment? How is this reflected in the staging/testing/development environments?

Supporting different Environments

- How do you construct your application such that you can move seamlessly between the different environments?
- How do you construct your application to support automated builds and tests?

Summary

- Certain Quality Attributes are more relevant than others for a Cloud Application
- You must first find your particular blend of quality attributes
- After this, designing a cloud application is similar to designing the architecture for any other system.
- The Execution View (and hence the Module View) plays a more significant role
  - Quite Obviously; this is the factor that has changed.
- Pay extra attention to *privacy*, and your development environment