

# The Challenges of Operating a Computing Cloud and Charging for its Use

Marvin Theimer

VP/Distinguished Engineer

Amazon Web Services

# Customers Want It All

- Lots of features and all the “ilities”
- Pay as little as possible
- Get it as soon as possible

# Trade-offs Must Be Made

- Inherent tension between customers' desires
- MUST work backwards from the customer
- It's not always obvious what each customer really wants

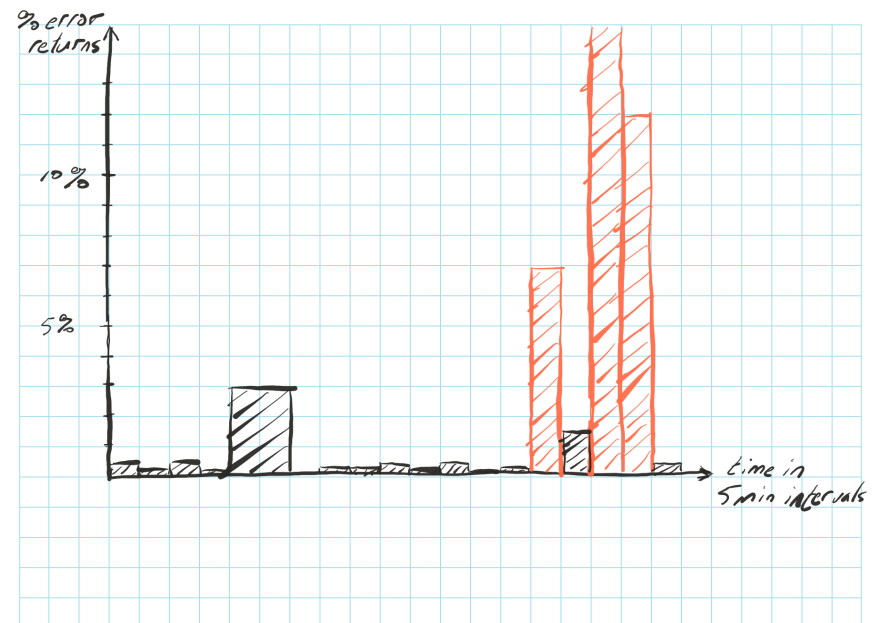
# Scaling Challenges

- A big compute cloud has at least a million physical servers world-wide
- Amazon S3 stores trillions of objects, contains exabytes of data, and fields millions of requests/second
- A service-oriented architecture (SOA) implies there are many services
  - Amazon has tens of thousands of services
  - A big service like Amazon S3 may require tens of thousands of servers

# You Need Availability Too

- Example definition of availability:

“The number of 5 minute intervals during which the ratio of error returns (http 500's) to total system requests is less than 5% over the total number of 5 minute intervals.”



# Levels of Availability

• Availability	Amount of down time per year
• 99.8%:	17.5 hours
• 99.9% (3x9's):	8.8 hours
• 99.99% (4x9's):	52.6 minutes
• 99.999% (5x9's):	5.26 minutes
• 99.9999% (6x9's):	31.5 seconds
• 99.99999% (7x9's):	3.15 seconds

# Some Implications of Various Levels of Availability

- 99.8%: 17.5 hours
  - You might cripple your business (e.g. Intuit on Apr 15<sup>th</sup>)
- 3x9's: 8.8 hours
  - You can afford to do occasional small scheduled down times
- 4x9's: 52.6 minutes
  - Can't do scheduled down times of any significance
  - Paged human has about 30-40 minutes to correct/restart things

# Some Implications of Various Levels of Availability

- 5x9's: 5.26 minutes
  - Paged human won't be on-line before you've exceeded your yearly SLA
  - De-facto need fully automated failure response system
  - Humans can only be involved with longer-term trends management
- 6x9's: 31.5 seconds
  - Have to redefine what you mean by availability (5 min. intervals too coarse)
  - In the range of throttling delays
- 7x9's: 3.15 seconds
  - Below the practical threshold for distributed leased locks

# Reality Bites

- Developers are fallible
- Cloud services evolve quickly
- Near-perfect automation/fault-tolerance is expensive
- Current state-of-the-art requires humans in the loop to deal with unforeseen circumstances
- You can build 6x9's available services, but it may not represent the right cost/benefit trade-off for most customers

# You Need Logging

- Volume of log traffic is measured in TB/hour
    - For a large service the log volume is still of that order of magnitude
  - Can't just grep it: you need a full-blown search capability
  - Richer queries imply even more technology
  - Need for timely answers pushes you towards near-real-time support
- 
- It's all technology feasible
  - It just costs a lot
  - How much cost is justifiable?

# You Need Metrics as Well as Logging

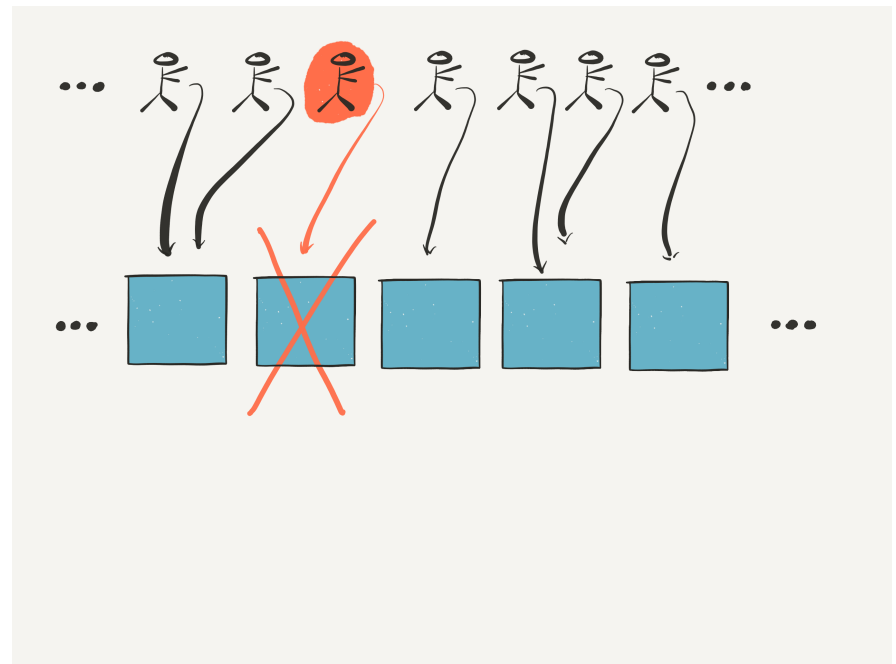
- Log queries are for debugging
- To determine whether a service/system is behaving properly you need metrics
- Ideally you track “everything”
- Far too expensive
  - Cost of gathering
  - Attention cost
- You have to figure out the metrics “working set” you need
- What are your “leading” metrics?

# Metrics Challenges

- The working set may change in unobvious ways as your service – or its workloads – evolve
- Important to have “trip wire” metrics
- Important to have automated alarms
  - Also need to have alarm deduplication/squelching

# Availability as Seen by Individual Customers

- A service can be 99.99% available and an individual customer can still have a really bad day
- Ideally, want near-real-time “top-N” metrics
- These are not cheap



# Latency as Seen by Individual Customers

- One definition of a latency SLA:

“The number of 5 minute intervals during which the ratio of returns with latency higher than the latency SLA to total system requests is less than 5% over the total number of 5 minute intervals.”

- Same “bad day” problem exists for latency as for availability
- Need to monitor p90, p99, p99.99, and even p100

# Developing, Testing, Deploying, Operating at Scale

- Amazon Web Services (AWS) launched O(1000) features last year. Customers are impatient for more
- The variety of workloads and exception scenarios (failures, distributed denial-of-service attacks, customer load spikes, etc.) is huge
- Increasing emphasis/demand for platform-wide features, such tagging, policy enforcement, etc.

# Things can change out from under you quickly

- Sudden load quantum leaps
  - Capacity challenge
  - Ramp-up challenge
- New features that have unintended scaling side effects
  - New feature in one place may accelerate the rate of load growth in another
  - Non-linear effects
  - Unintended consequences due to unexpected uses

# Testing is Crucial

- Avoiding the death spiral of mean-time-to-failure > mean-time-to-repair
- Testing: the only “truth” you have is what you test regularly
  - Regression tests
  - Scaling/performance tests
  - Fault tolerance tests
- The importance of testing to failure
  - Load testing to the breaking point along all relevant dimensions
  - Chaos monkeys
  - LSE tests (chaos armies and game days)

# You Can't Anticipate Everything

- Need rolling deployments
- Need (automated) rollback capability
- Root-cause analysis is challenging when “everything” is constantly in flux
  - Use CI/CD: lots of small, incremental changes are easier to deal with than a few “big bangs”

# Operational Readiness is Crucial

- Modeling your system
  - Security threat model
  - Failure model, including LSE analysis
- Operational readiness review (ORR) checklist
- On-call rotation
  - Primary personnel
  - Well-defined escalation paths, including to other services
- On-call run books
  - Have to be easy to understand and use
  - Must practice using them

# Humans in the Loop

- Humans are necessary because systems are
    - Extremely complex
    - Evolve at a ferocious rate
    - Exhibit difficult-to-anticipate emergent behaviors
    - Behave in non-linear ways
  - Humans are a huge problem because they are imperfect – especially at repetitive tasks
    - Multi-step standard operating procedures (SOPs) are a good source of errors
    - Ditto for cut-and-paste tasks
    - Ditto for complex, difficult-to-parse, text commands
- ➔ Need canned procedures that have simple invocation semantics

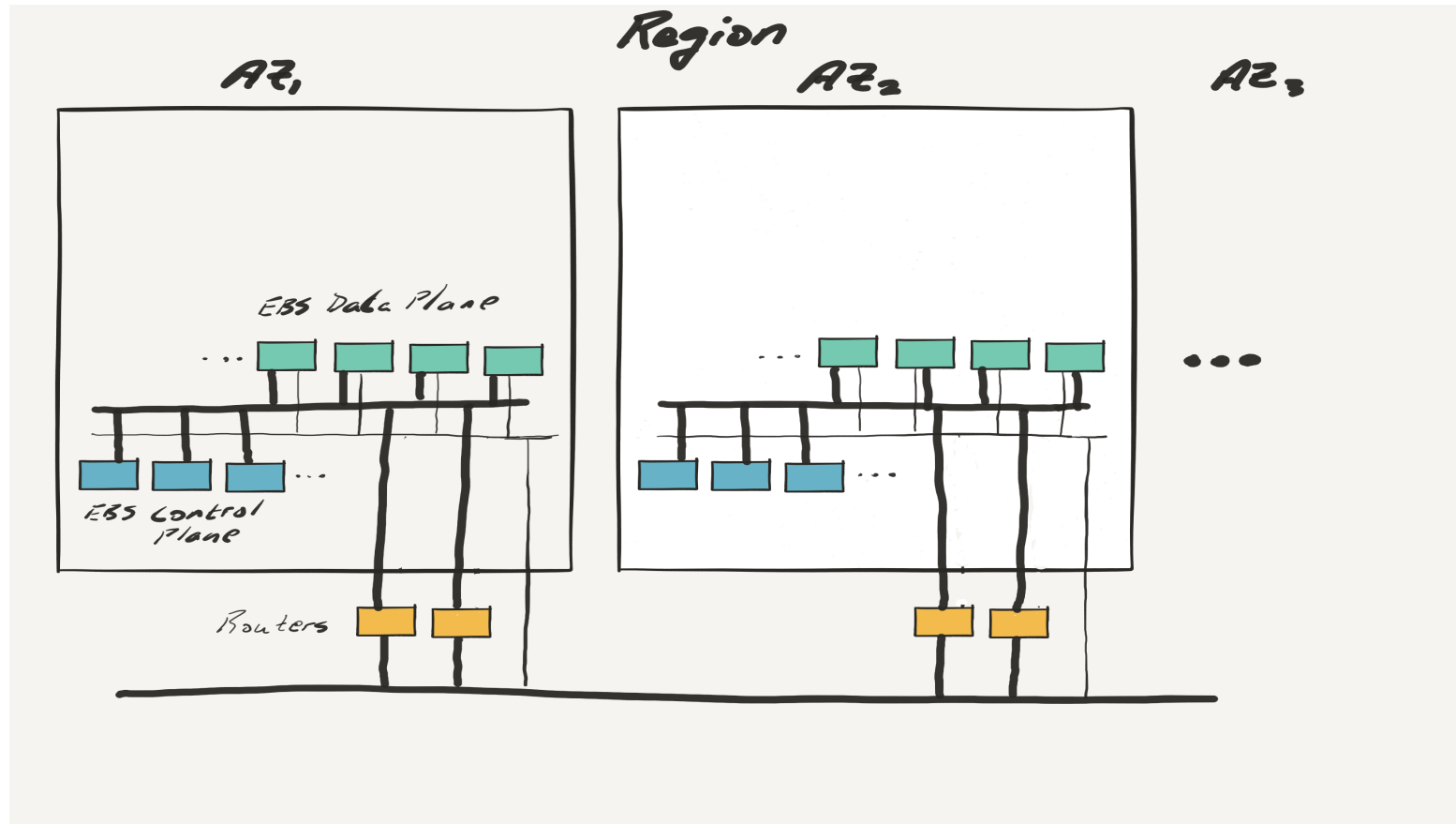
# The Tension Between Power/Efficiency and Safety

- Tools and APIs should be safe to use:
  - Projected outcome of an action should be clearly discernible
  - Ideally actions can be undone if necessary
- Safety adds friction
  - Danger of people inventing short-cuts
  - What's the “right” amount of safety friction to impose?
- Sometimes you need a power tool that will let you do “heart surgery”
  - How often do you use it?
  - How often do you practice with it?

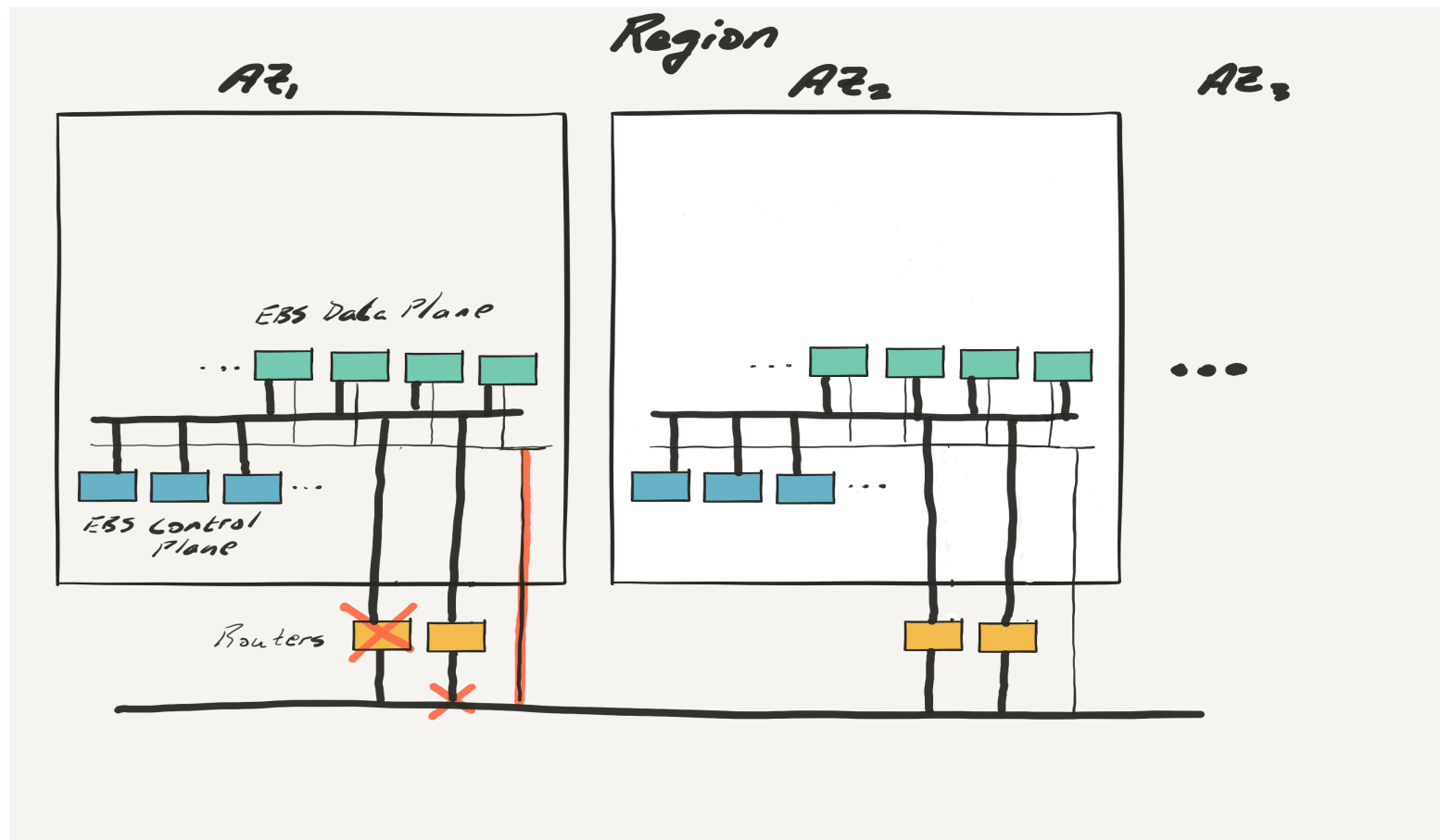
# “Correction of Error” Reports

- The importance of recording and propagating things learned
- Root cause analysis: “The 5 Whys”
  - Example: 2011 Amazon EBS outage
    - Network misconfigured during an upgrade
    - Re-mirroring storm
  - What’s the root cause?
    - Service control plane problems → Service data plane problems → network traffic problems → network misconfiguration → difficult-to-use tools for configuring network routers
- The importance of closed-loop action mechanisms vs. good intentions

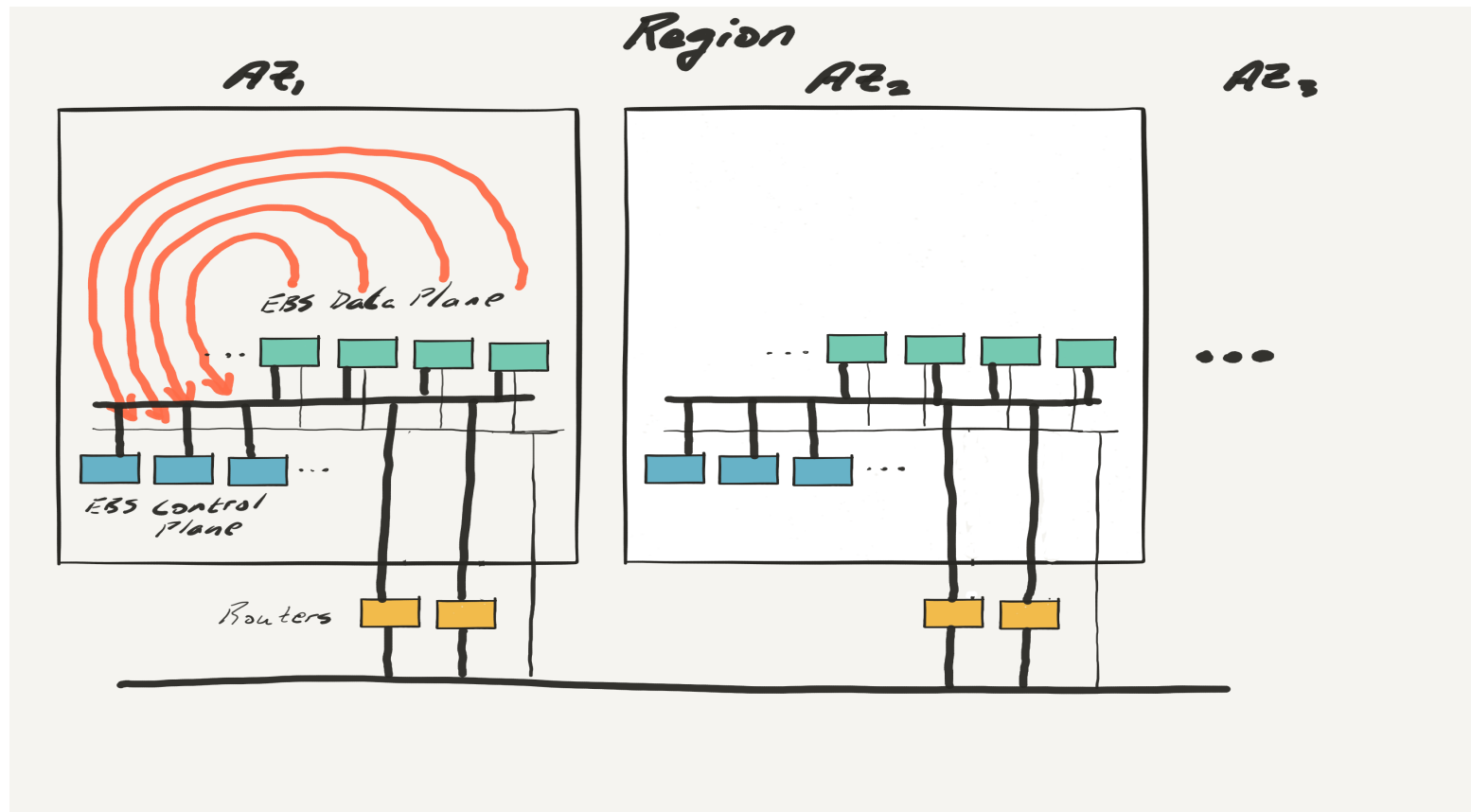
# Abstract Representation of EBS in a Region



# Initial Failure Event



# Follow-on Problems



# Charging for Use

- You have to build in support from the beginning (like with security)
  - Have to be able to track customers' usage along all relevant dimensions and across all backend systems and services
  - Metering volumes (at the edge) are measured in millions of records/sec and TB/hour.
- What's the right pricing model?
  - Fully cost-following models are very complicated
  - Simpler models may have unintended consequences

# Some Pricing Nuances

- Free tiers
  - Invitation to use
  - Also a simpler pricing model for “glue” resources
- Derivative usage
  - Resource usage enabled by other resource usage
  - Example: cheaper data ingestion leads to more compute

# Limiting Mistakes and Fraud

- High elasticity enables the ability to do a lot of damage quickly
- How do you distinguish legitimate requests for more resources from mistakes and fraud?
- Have to put dynamic limits on what can be used
  - Simplest – and least customer friendly – solution is universal soft quotas
  - Can make a quota customer-specific
    - Trusted customers get higher default limits
    - Past history used as predictor of future behavior
  - Differing payment strategies

# Summary and Conclusions

- A fundamental tension: customers want
  - rich feature set and capabilities
  - relentless cost reduction
  - everything as soon as possible
- At scale it's all about the tail
- Testing and automation are crucial, but the human (so far) still has to be in the loop – for better and worse
- What to charge has many nuances and requires support from the beginning