CMPE 282 Cloud Services
Cloud-Native Application Design Patterns

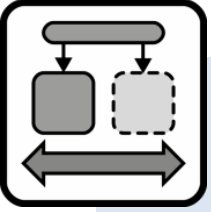
Instructor: Kong Li

Content

- What and why
- Coffee Shop
  - Decomposition
  - Workload
  - Data/state
  - Component refinement
  - Elasticity and Resiliency
- Composite Cloud Applications

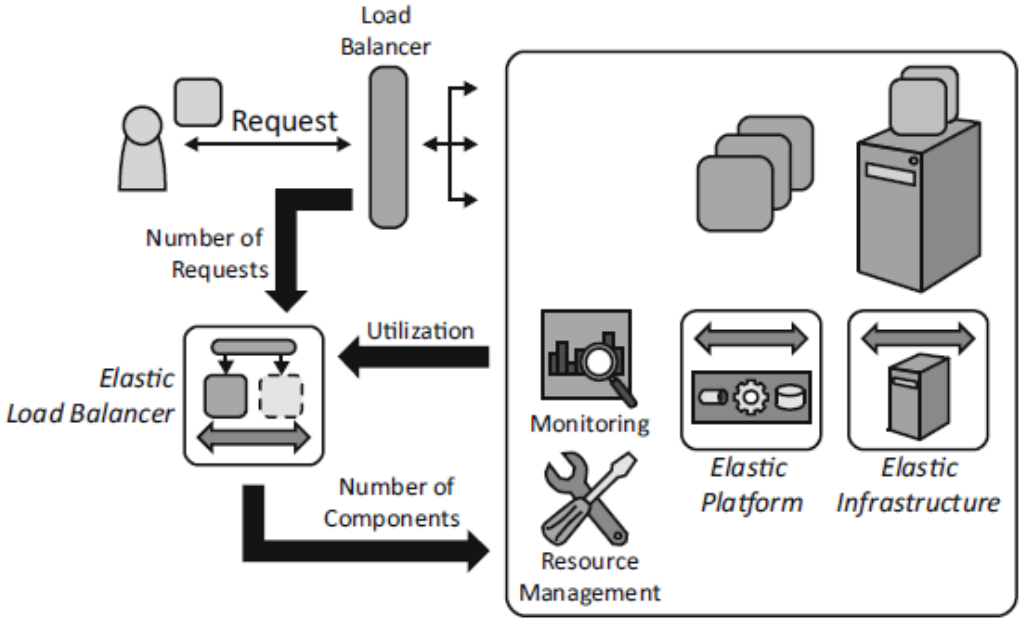
Coffee Shop App Design (5)

- Decomposition: How to distribute Application Functionality?
  - Distributed App
- Work load: What workload do components experience?
  - Static
  - Periodic
  - Once-in-a-lifetime
  - Unpredictable
  - Continuously Changing
- Data (State): Where does the application handle state?
  - Stateful
  - Stateless
  - Strict consistency
  - Eventual consistency
  - Data Abstractor
- Component Refinements: How are components implemented?
  - Message-oriented Middleware
  - User Interface Component
  - Processing Component
  - Batch Processing Component
  - Multi-component Image
- **Elasticity and Resiliency**
  - **Elastic Load Balancer**
  - **Elastic Queue**
  - **Node-based Availability**
  - **Environment-based Availability**
  - **Watchdog**



Elastic Load Balancer

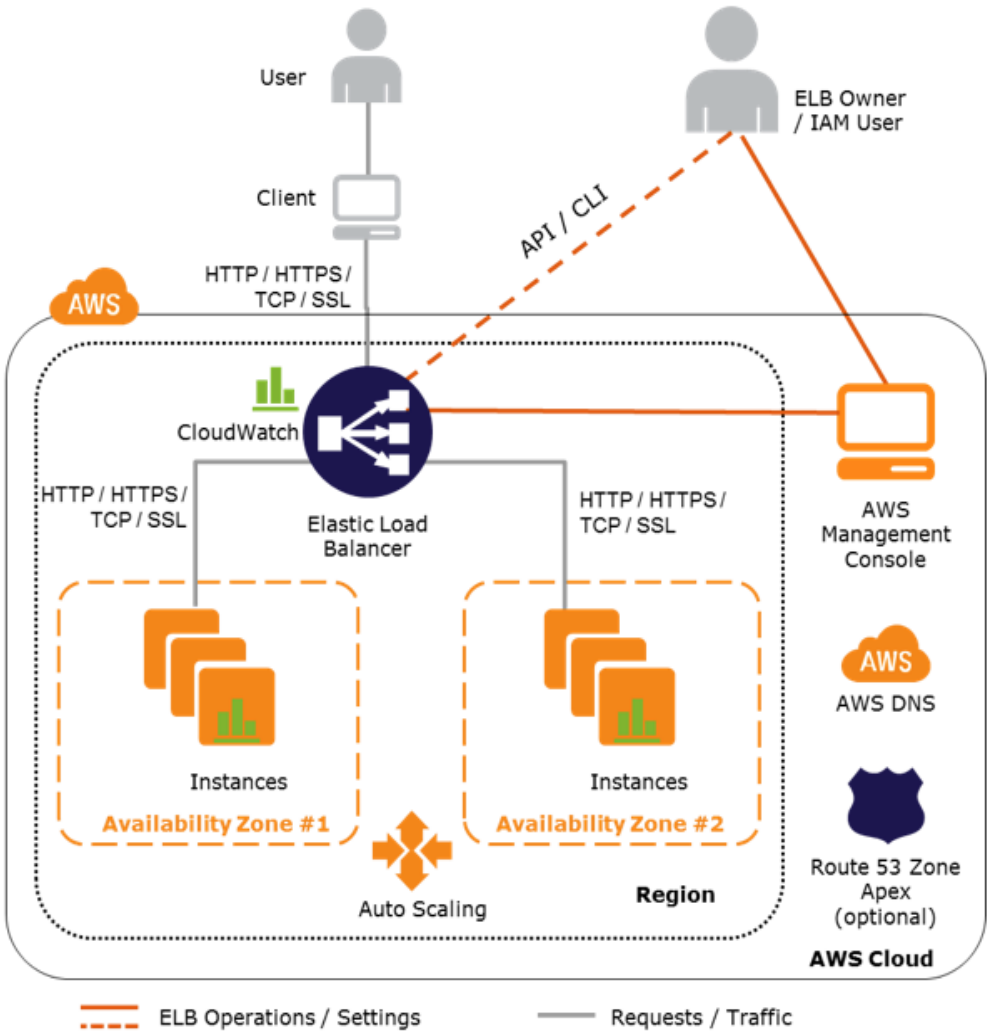
- Intent: # of **sync** accesses to an elastically scaled-out app and its util% is used to determine # of required app component instances



- AWS ELB, auto scaling
- Google GCE auto scaler, GAE auto scaling
- Azure Traffic Manager, Cloud Services auto scale

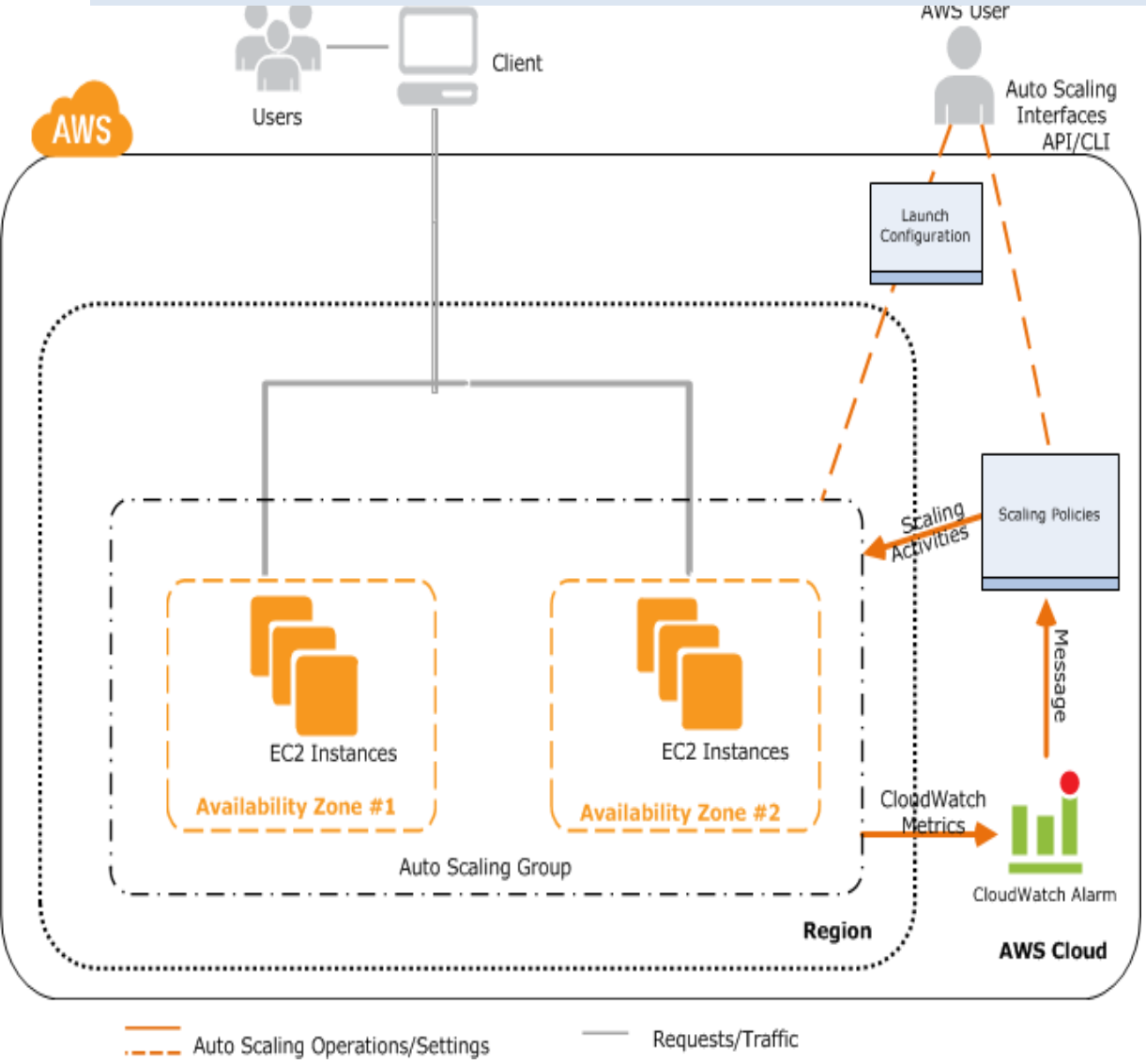
- Solution: Based on # of sync user requests and/or util%, the elastic load balancer determines # of component instances, and performs provisioning and decommissioning on elastic platform or elastic infrastructure
  - Horizontal scale: scale out / scale in
  - Network load balance and workload load balance

AWS - Elastic Load Balancing (ELB)



- Distribute incoming apps traffic across AZs (each w/ multiple EC2 instances)
  - *Classic LB: Network* load balance
    - to healthy EC2 only
  - *Application LB*: *content-based* routing
    - across > 1 services or containers on EC2 instances
  - Integration w/ auto scaling (workload LB)
- vs. VMware DRS

AWS - Auto Scaling

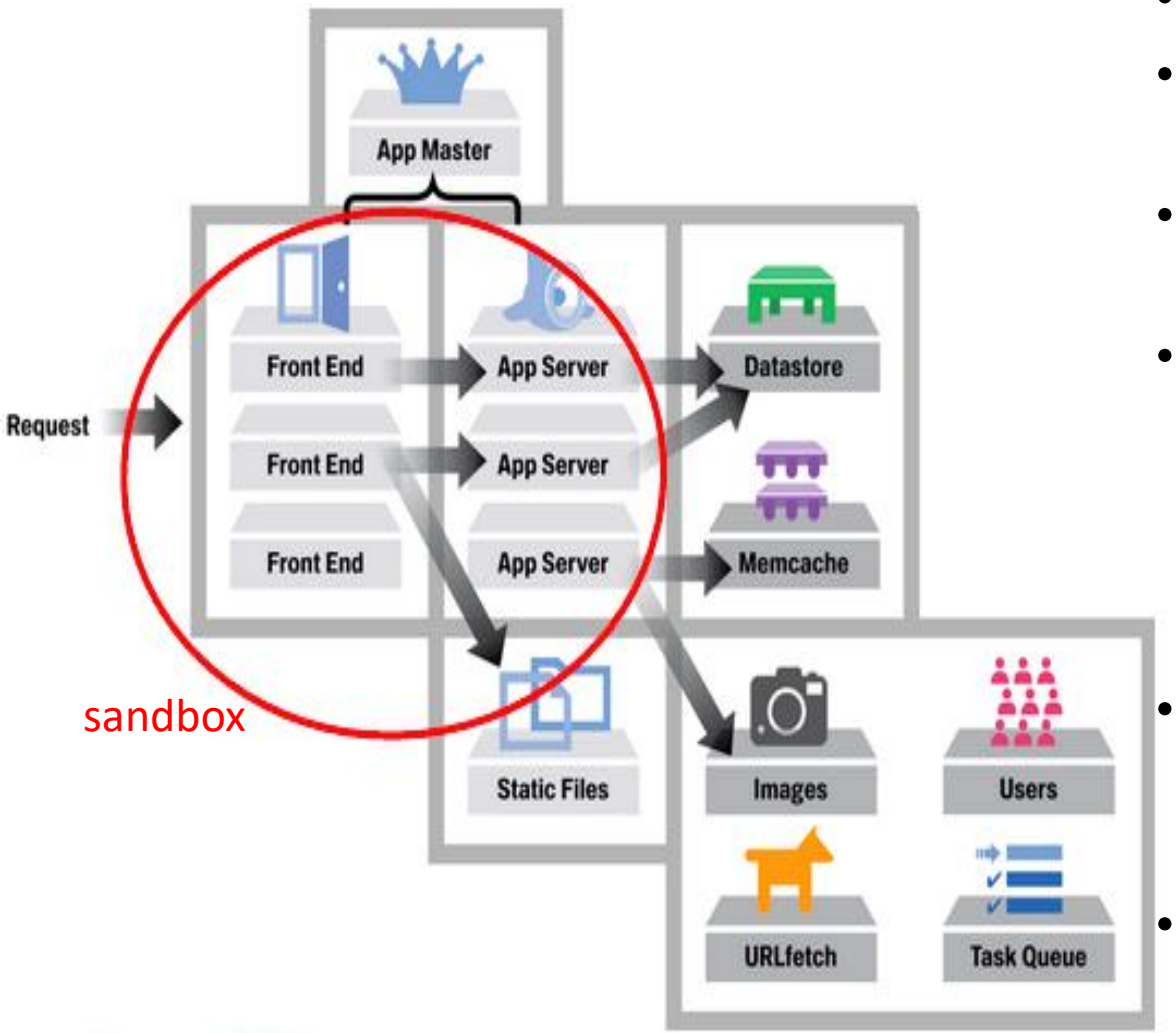


- Auto scale out/in
  - *Workload* (resource usage) load balance
  - Keep "optimal" # of VMs at optimal util% across AZs
  - Auto scaling group
  - Policies
    - Launch config
    - Min/max group size
    - AZs
    - conditions → alarms → actions
- Enabled by CloudWatch
- In cooperation with ELB
- ≡ vRealize Orchestrator
- vs. VMware DRS

Google Compute Engine (GCE)

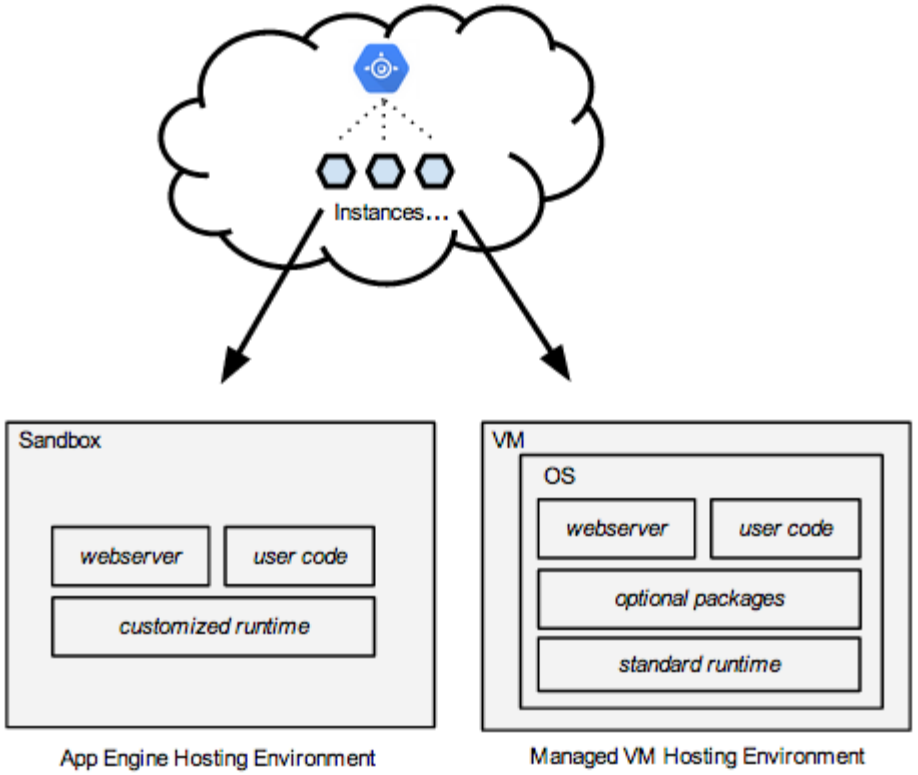
- IaaS: KVM-based VM
- OS: various Linux, Windows
- Machine types
  - Standard, small
  - High CPU, high memory
- Instance template
- Load balancing: Replica pool
  - Network, HTTP (cross-region)
- Instance group: pool of VMs
  - Size up/down
- Autoscaler: managed instance group
  - Network, HTTP (cross-region)
  - CPU util%, Cloud metrics
- Authorizing to/from other Google service: OAuth 2.0
- Resource: global / region / zone
  - Zone: isolated location in a region
  - Region-specific resources can be accessed by resources in the region
  - Isolation of failure, redundancy
- Disk snapshot: FS only; no VM memory; VM up or down
- no VM hot clone
- Host maintenance policies
  - [default] VM live migration
  - Terminate and (optionally) restart
- VM crash: [default] auto restart

Google App Engine (GAE) - Standard



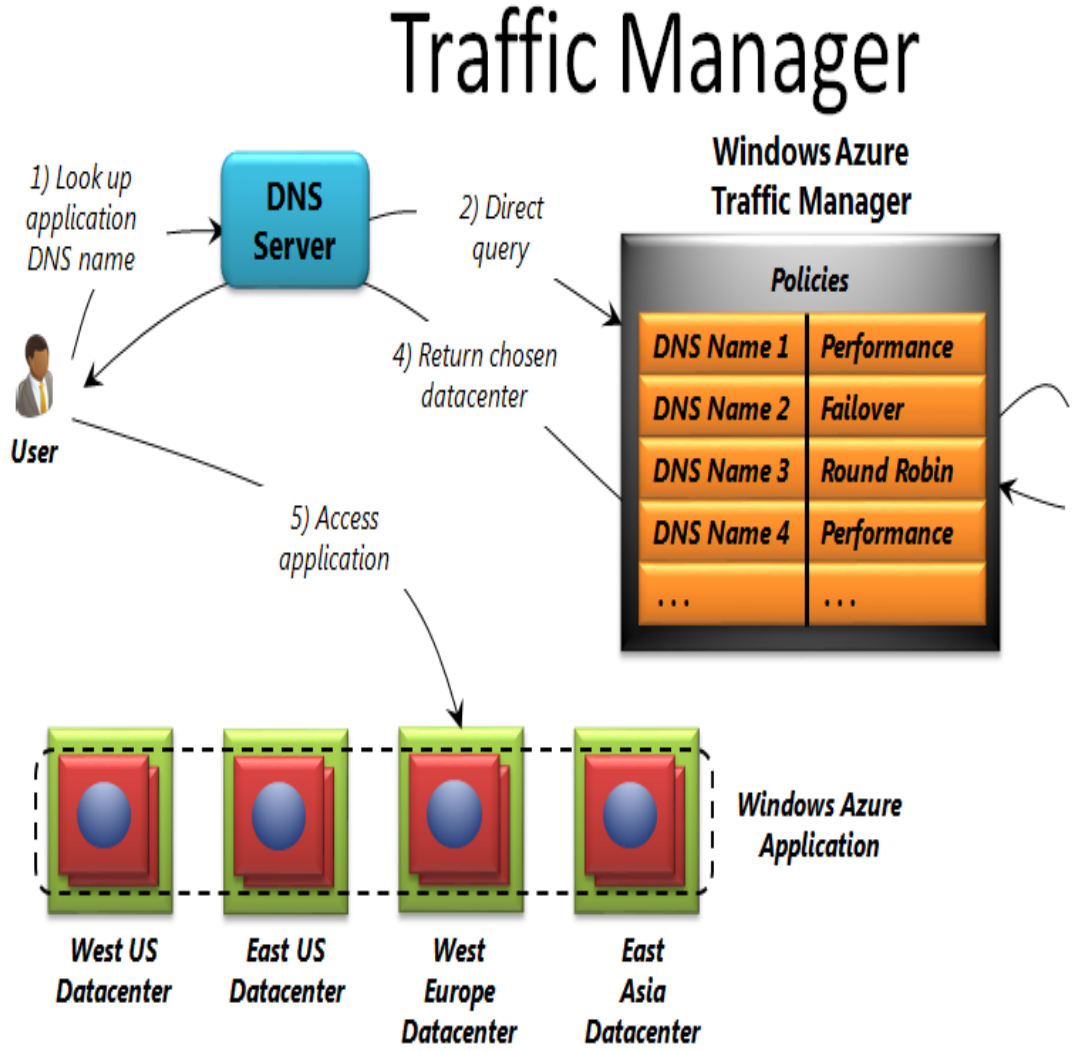
- PaaS: web app
- Sandboxed (container-based) env: Python, Java 7, PHP, GO
- Instance scaling: auto (req rate, latency), basic, manual
- Storage
  - Datastore: NoSQL
  - Cloud SQL: MySQL
  - Blobstore
  - Cloud Storage
- Communication
  - Task queue: async process
  - Endpoints: client lib/REST
- Scheduled tasks for triggering events

Google App Engine (GAE) - Flexible



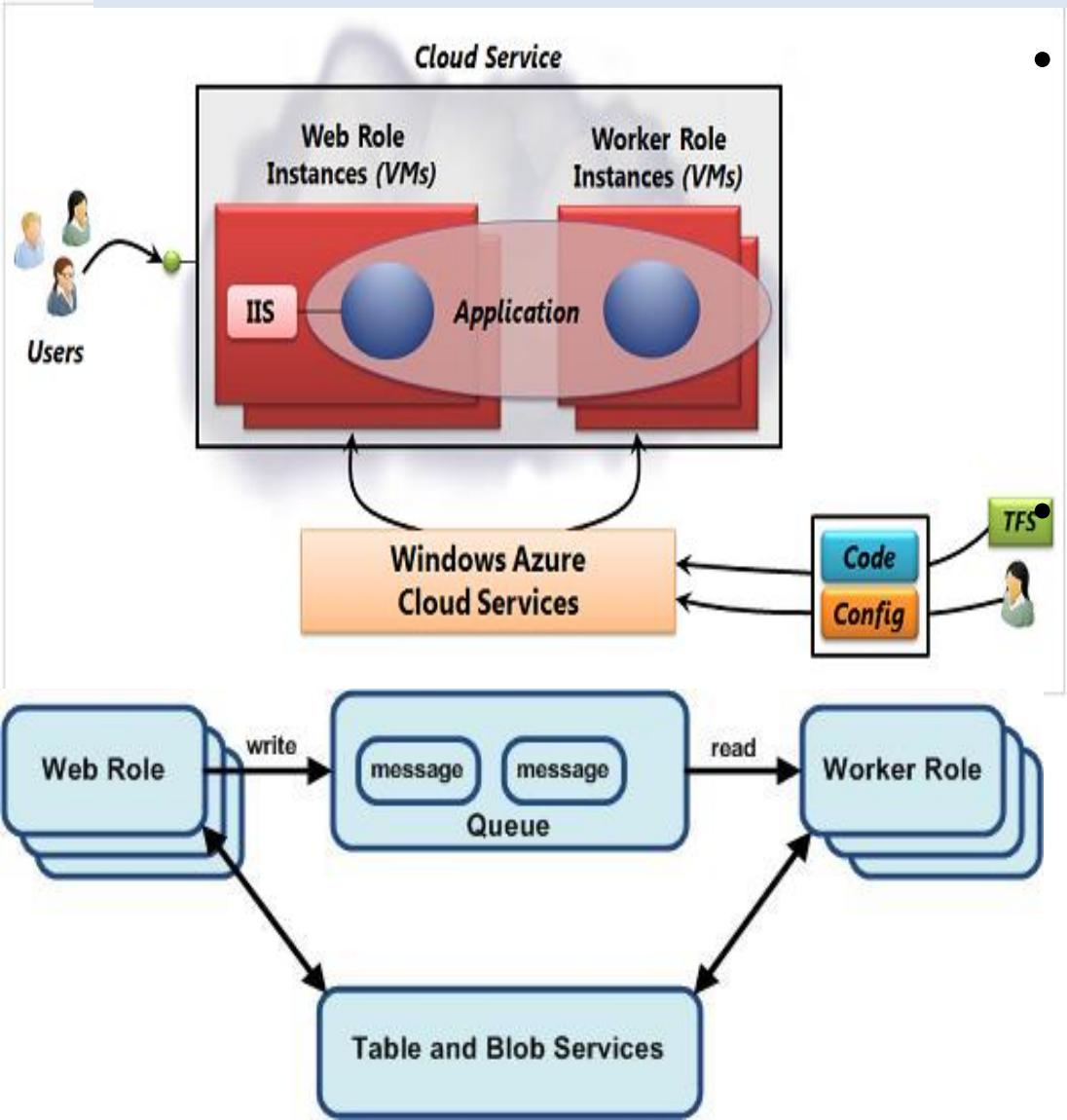
- Initial name: Managed VMs
- run App Engine apps on configurable GCE (VMs)
  - More cpu / mem options
  - User configurable custom runtimes
  - Java 8, Python, Ruby, GO,Node.js, (Docker-based) custom runtime
  - Apps: mix-and-match standard and flexible env
- Auto-scaling
- Load balancing

Azure - Network Services - Traffic Manager

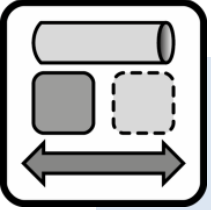


- Load balance incoming traffic for performance and HA
  - Distribute traffic over multiple locations
- Monitor health status
- Load balancing options
  - Failover
  - Round robin
  - Performance

Compute - Cloud Services

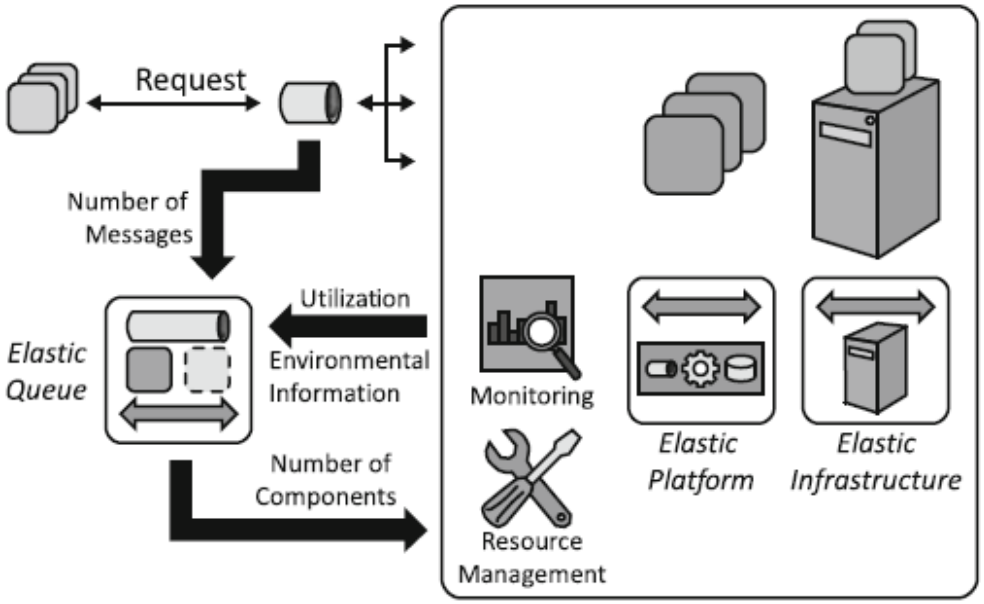


- PaaS for apps
  - VM roles: web role, worker role
  - Monitor: failure detection (apps, VM, host) & restart
  - Auto scale, load balancing
  - HA & FT: VM pool
- Cloud service apps
  - Do not create VM directly (PaaS)
  - User deploy apps and specify # of web/worker roles
  - apps state: only in SQL DB, blobs, tables, etc. Not in file systems of VM (why?)



Elastic Queue

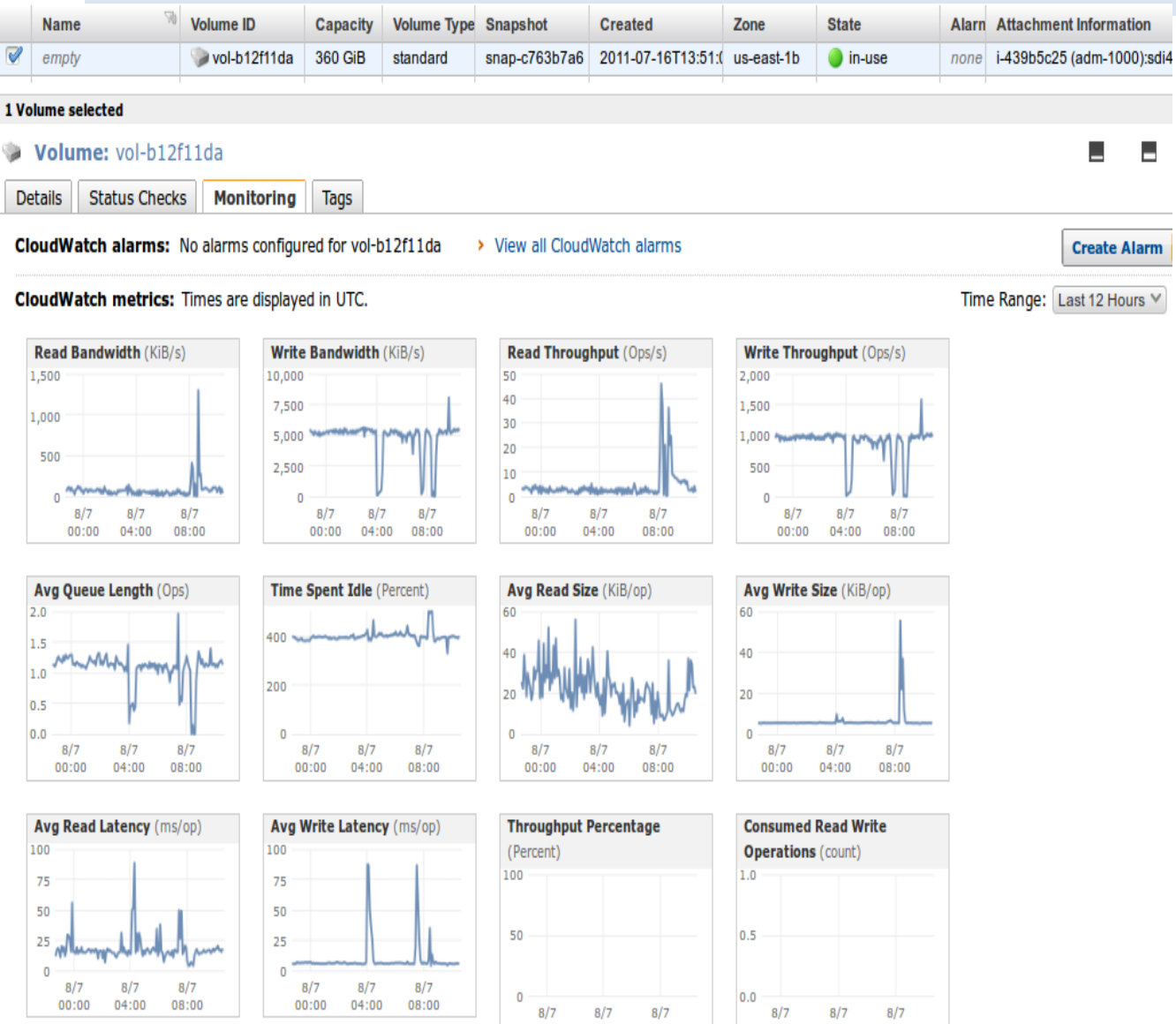
- Intent: # of **async** accesses via messaging to an elastically scaled-out app and its util% is used to adjust # of required app component instances



- AWS CloudWatch

- Solution: the elastic queue monitors the queue (that distributes msg to component instances) and/or util% and determines # of component instances, and performs provisioning and decommissioning on elastic platform or elastic infrastructure
  - Horizontal scale: scale out / scale in
  - Network load balance and workload load balance

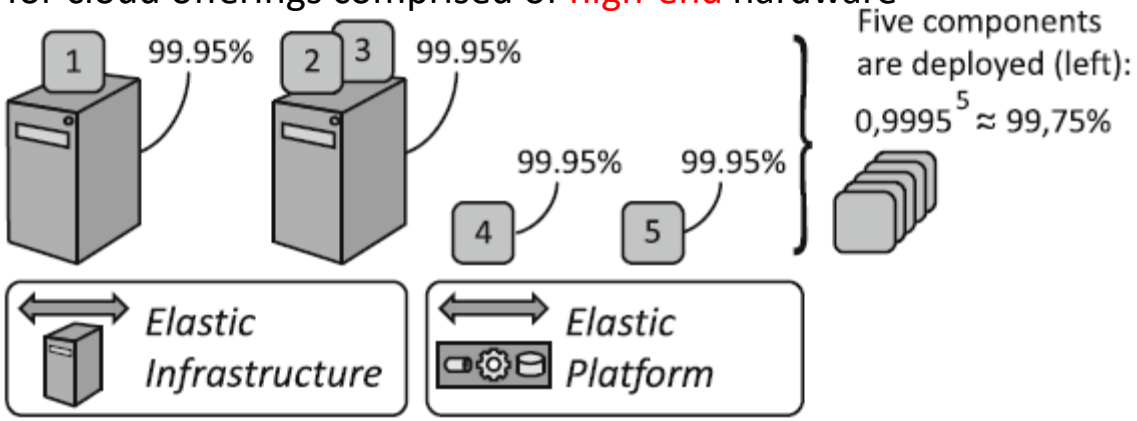
CloudWatch



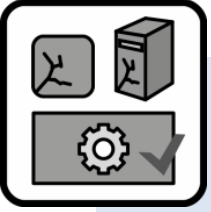
- Mgmt: monitor AWS cloud resources and apps
  - Standard metrics
  - Custom metrics (from customer's apps and services)
- Policy-based *Auto Scaling*
  - Auto scaling group
  - Add/remove (EC2) instances dynamically based on metrics
    - util%
    - queue length
    - etc.

Node-based Availability

- Intent: A cloud provider guarantees the availability of *individual* nodes, such as individual virtual servers, middleware components, or hosted app components
  - often used for cloud offerings comprised of high-end hardware

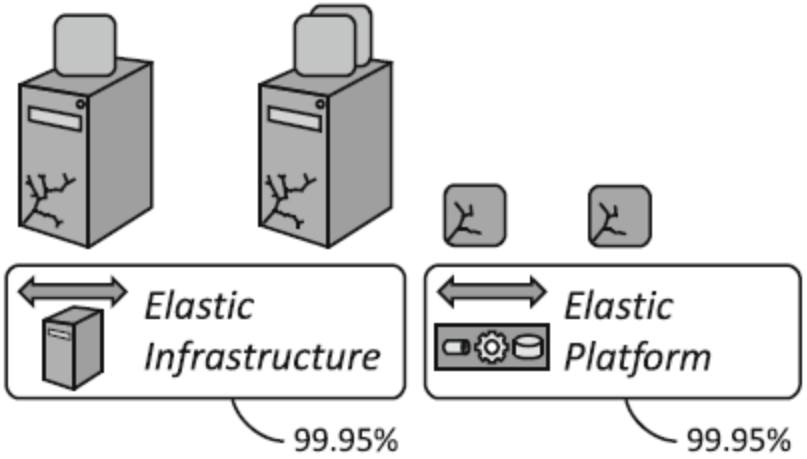


- Solution: The provider assures availability for *each individual* app component (elastic platform), or the provided virtual server (elastic infrastructure)
  - How to monitor? Util%, test data/msg, heartbeat, etc.
  - customer estimates overall avail% = ($node_1$ avail%) * ($node_2$ avail%) * …
  - If assured avail% is low, customer may incorporate redundancy and failure replacement on the app level (deploying multiple app component instances)



Environment-based Availability

- Intent: A cloud provider guarantees the (overall) availability of the env hosting individual nodes, such as virtual servers or hosted app components
  - often used for cloud offerings comprised of commodity hardware

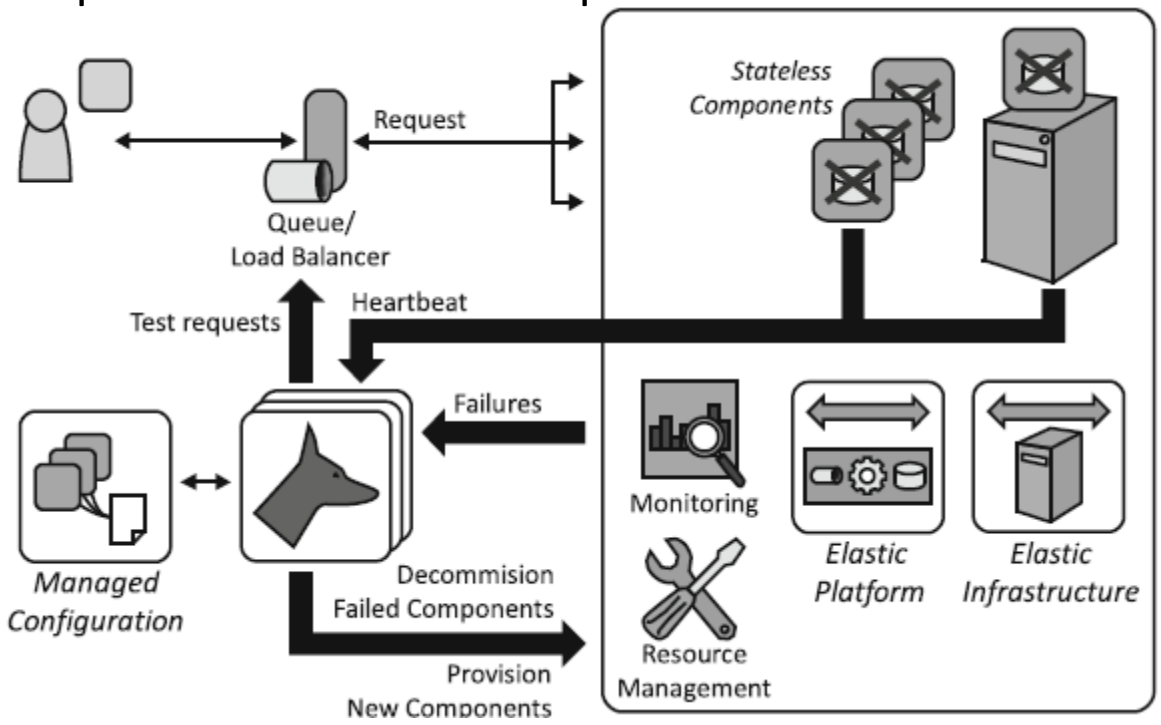


- AWS SLA

- Solution: The provider assures the *overall* availability for the provided env
  - no notion of availability for individual app components or virtual servers
  - customer incorporates failures in app architectures and runtime mgmt
  - AWS SLA: unavailability = {all running instances cannot be reached > 5 min and no replacement instances can be provisioned}
    - Deploy redundant instances to multiple “availability zones”

Watchdog

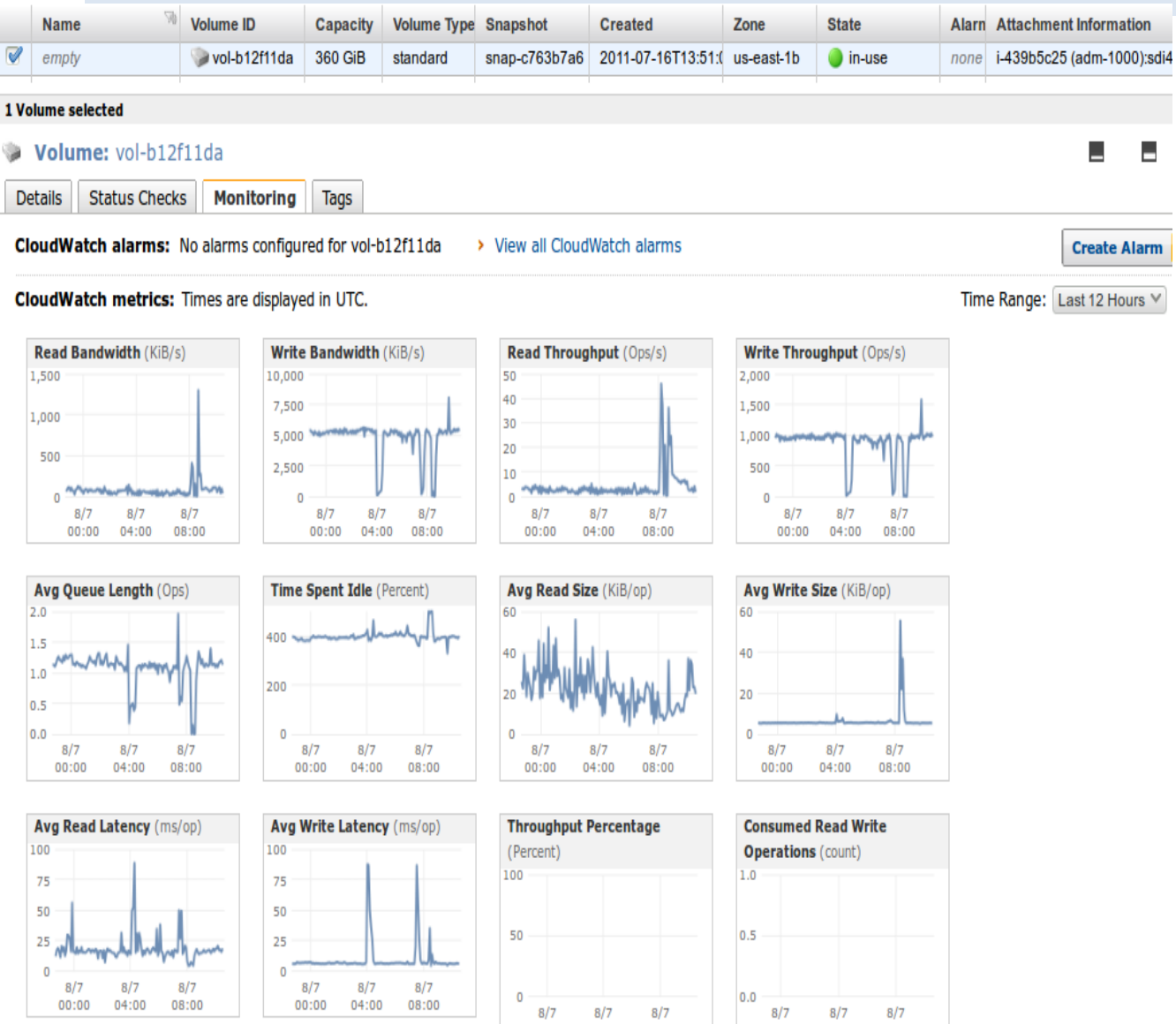
- Intent: HA applications cope with failures by monitoring and replacing app component instances if the provider-assured availability is insufficient



- AWS CloudWatch
- RightScale
- Scalr

- Solution: Scale-out *stateless* components (virtual servers and/or app components) are monitored by a watchdog component and replaced in case of failures
  - Watchdog itself must be implemented as HA

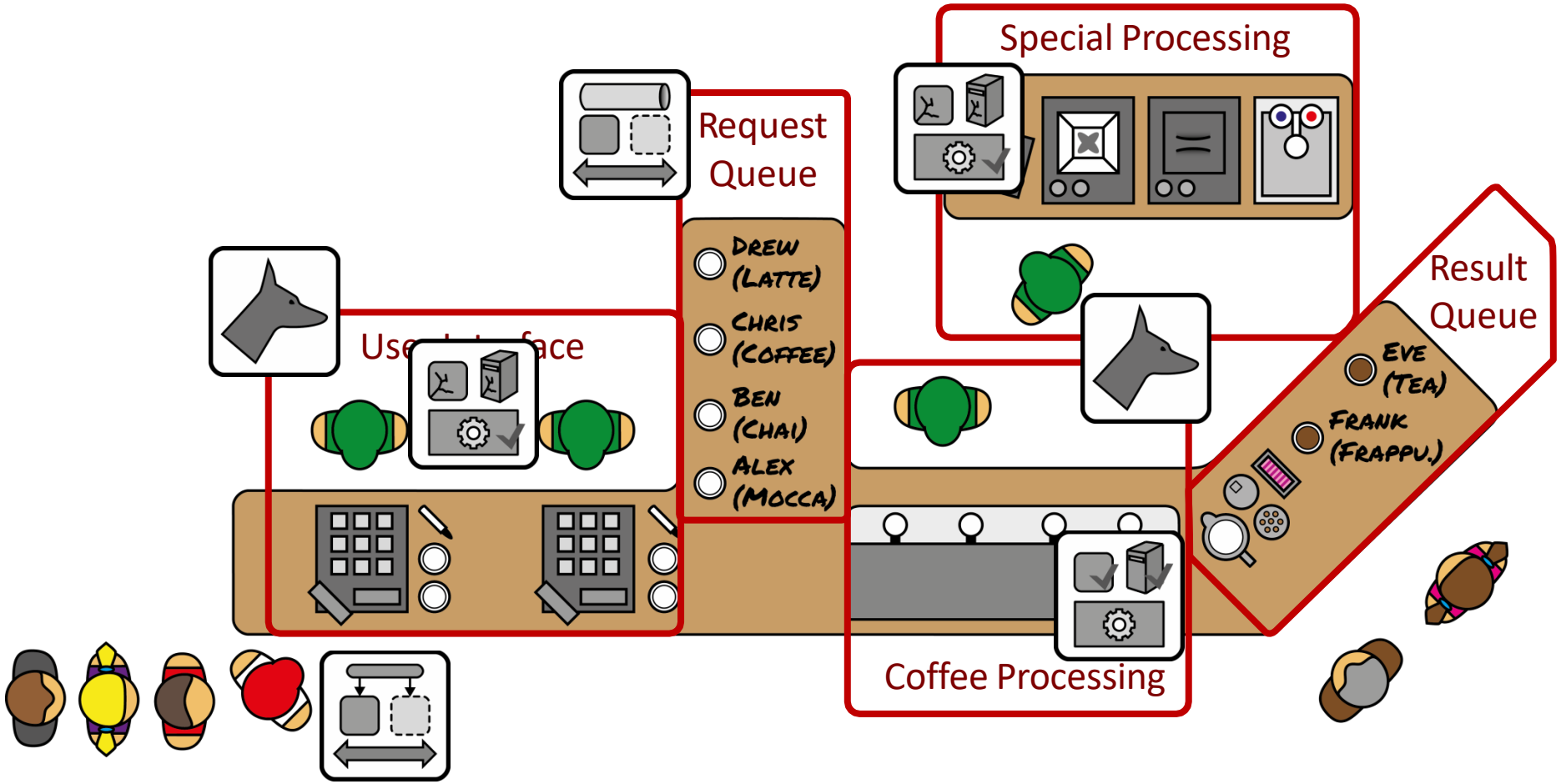
CloudWatch



- Mgmt: monitor AWS cloud resources and apps
  - Standard metrics
  - Custom metrics (from customer's apps and services)
- Policy-based *Auto Scaling*
  - Auto scaling group
  - Add/remove (EC2) instances dynamically based on metrics
    - util%
    - queue length
    - etc.

Coffee Shop - Elasticity and Resiliency

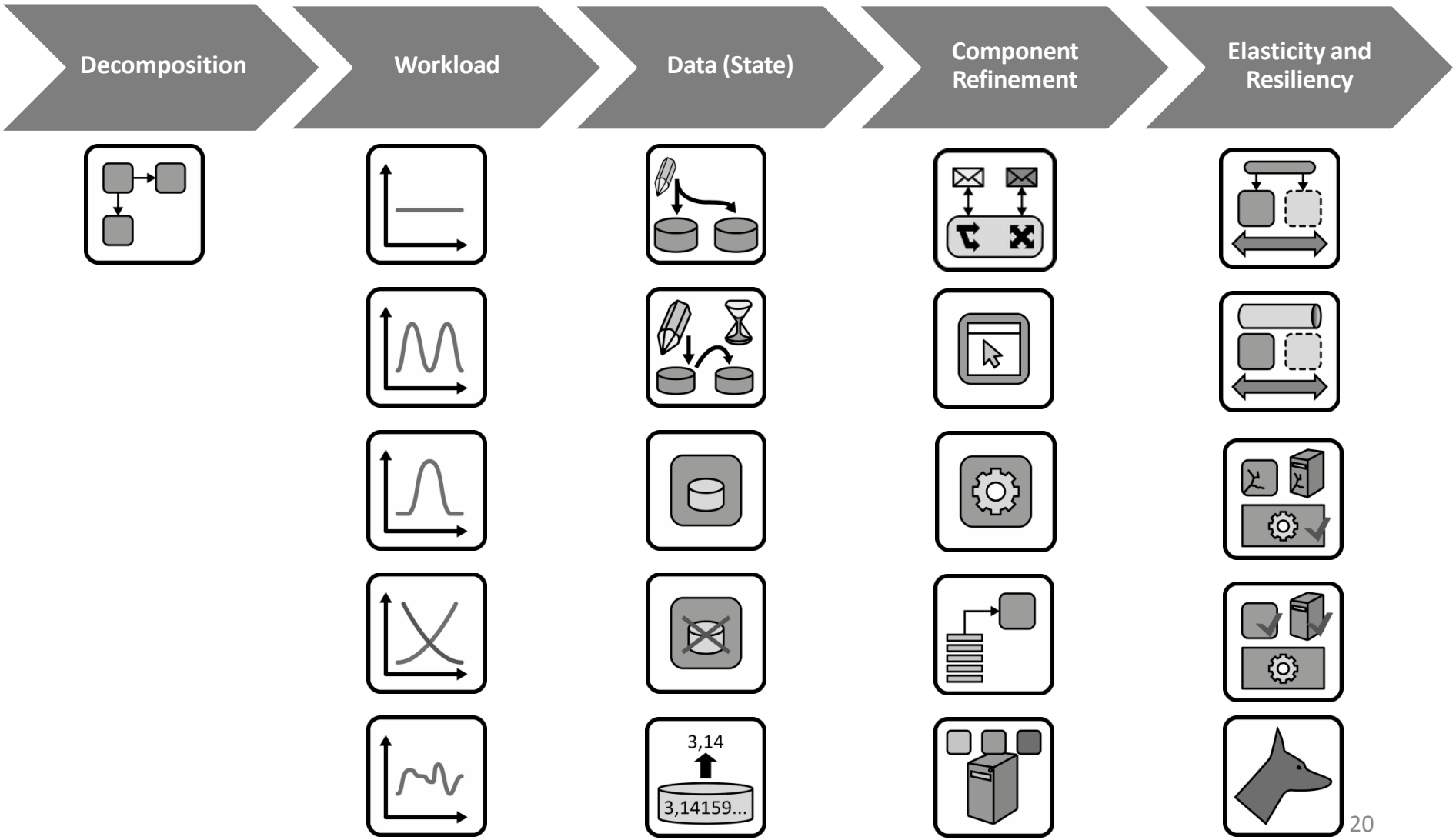
What shall happen if workload changes or something fails?



Lessons - Elasticity and Resiliency

- Analyze availability assured by provider (node-based or environment-based)
- In case of low node-based availability and environment-based availability implement a watchdog
- Use msgs and reqs to determine necessary component instances

Coffee Shop - Summary

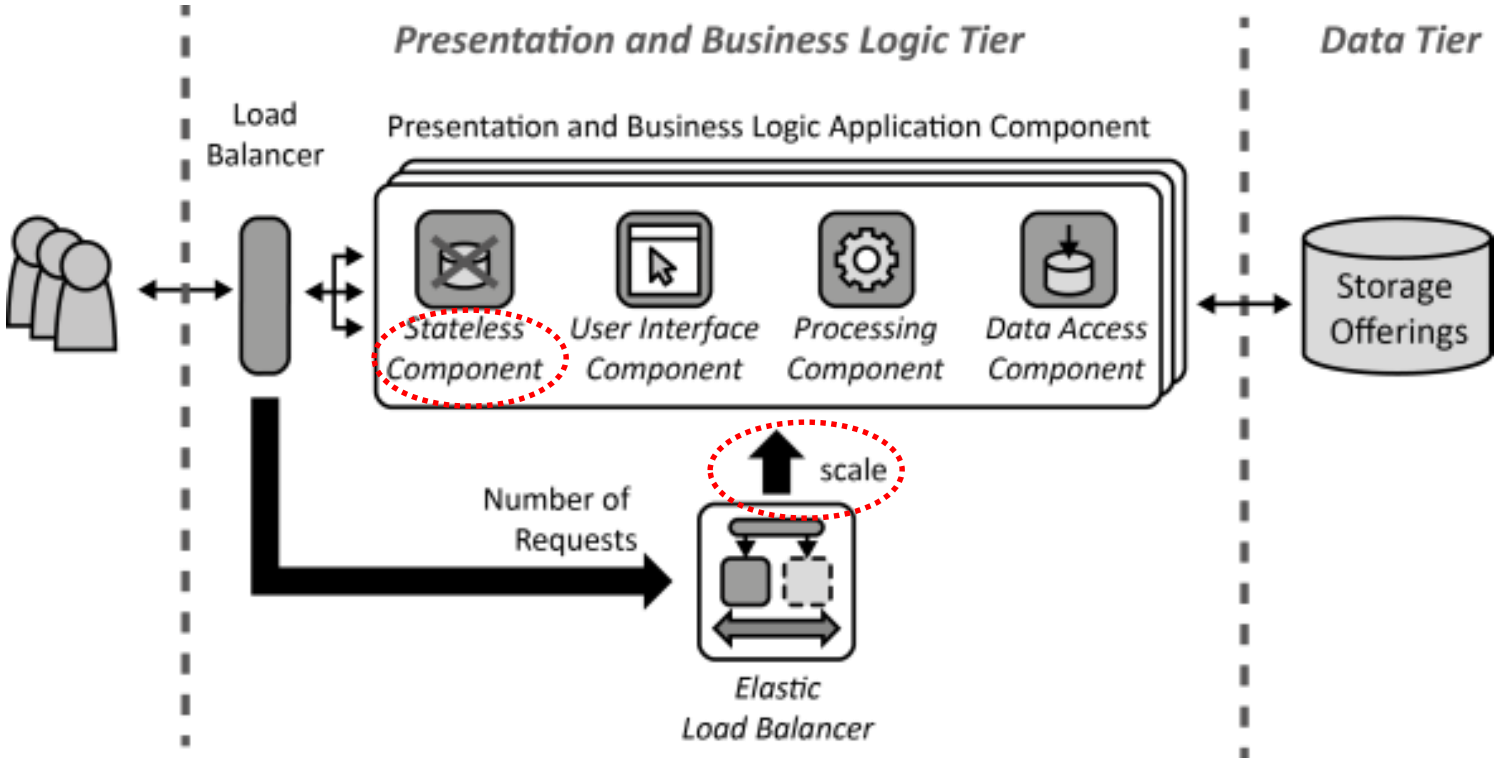


Composite Cloud Applications

- Native Cloud Applications
  - Two-Tier Cloud Application
  - Three-Tier Cloud Application
  - Content Distribution Network
- Hybrid Cloud Applications
  - Hybrid User Interface
  - Hybrid Processing
  - Hybrid Data
  - Hybrid Backup
  - Hybrid Backend
  - Hybrid Application Functions

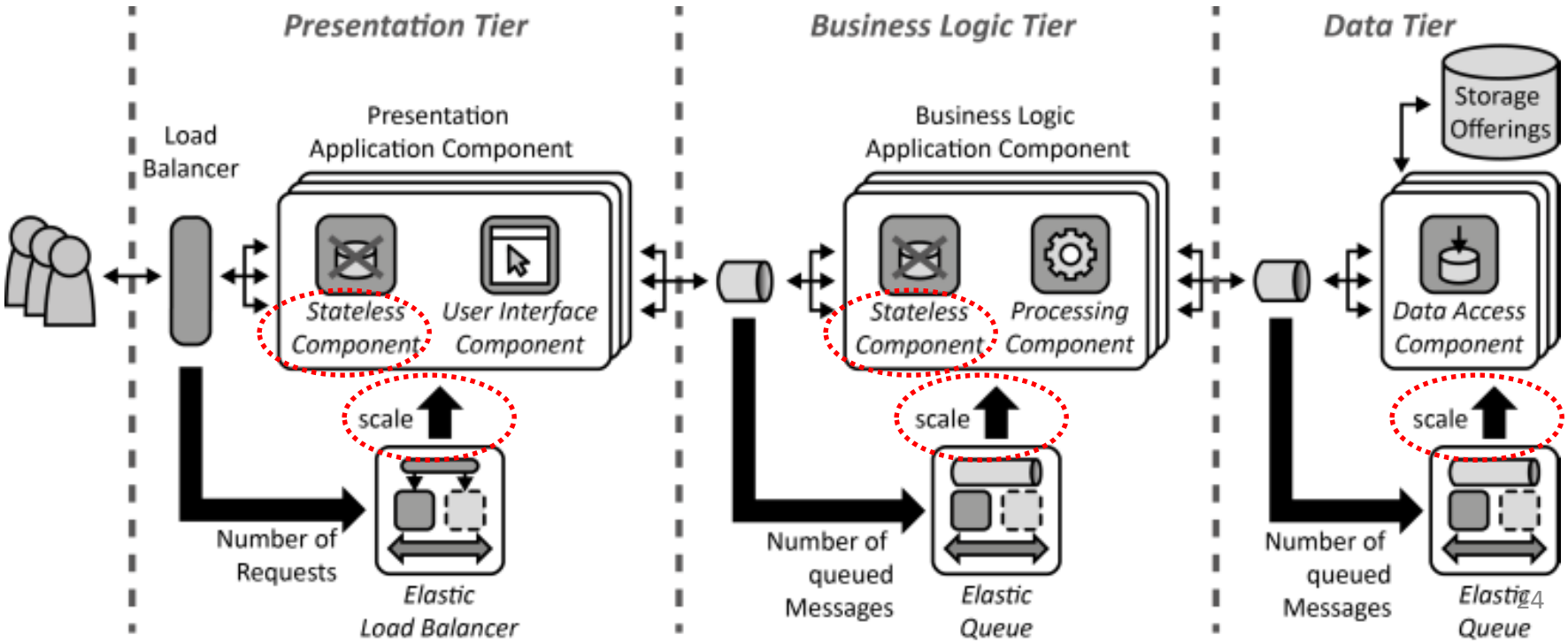
Two-Tier Cloud App

- Intent: Presentation and business logic is bundled to one stateless tier that is easy to scale. This tier is separated from the data tier that is harder to scale and often handled by a provider-supplied storage offering



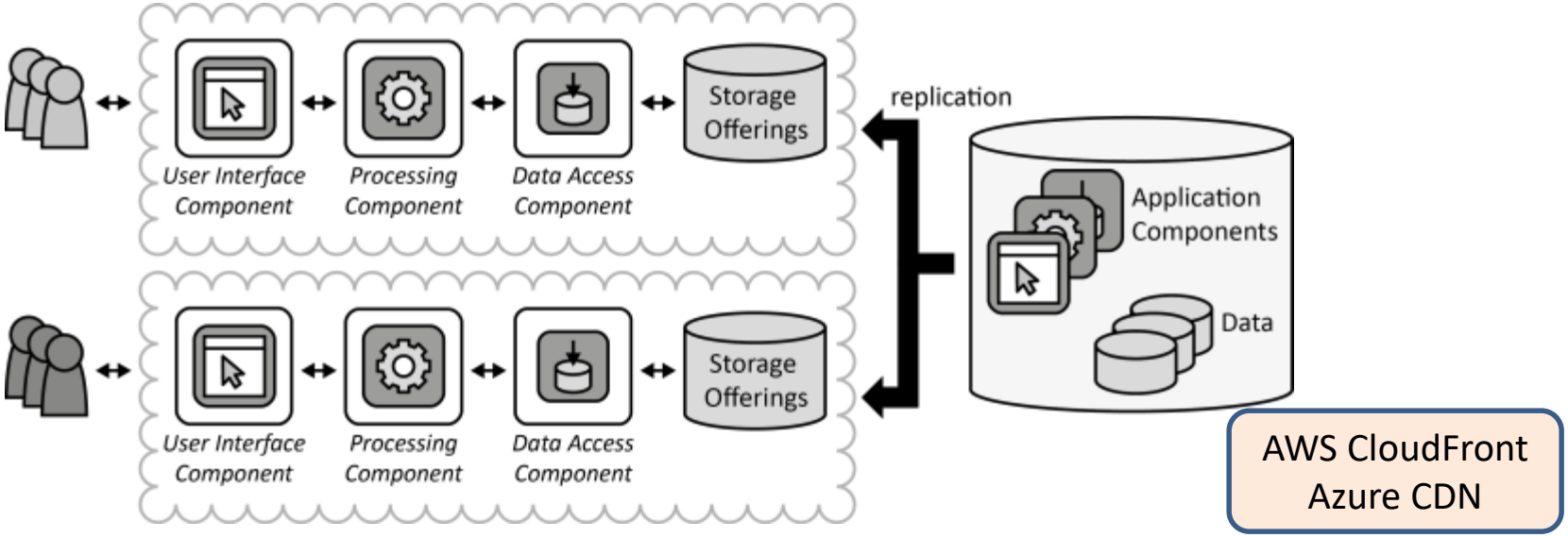
Three-Tier Cloud App

- Intent: The presentation, business logic, and data handling is realized as separate tiers to scale stateless presentation and compute-intensive processing independently of the data tier, which is harder to scale and often handled by cloud provider



Content Distribution Network

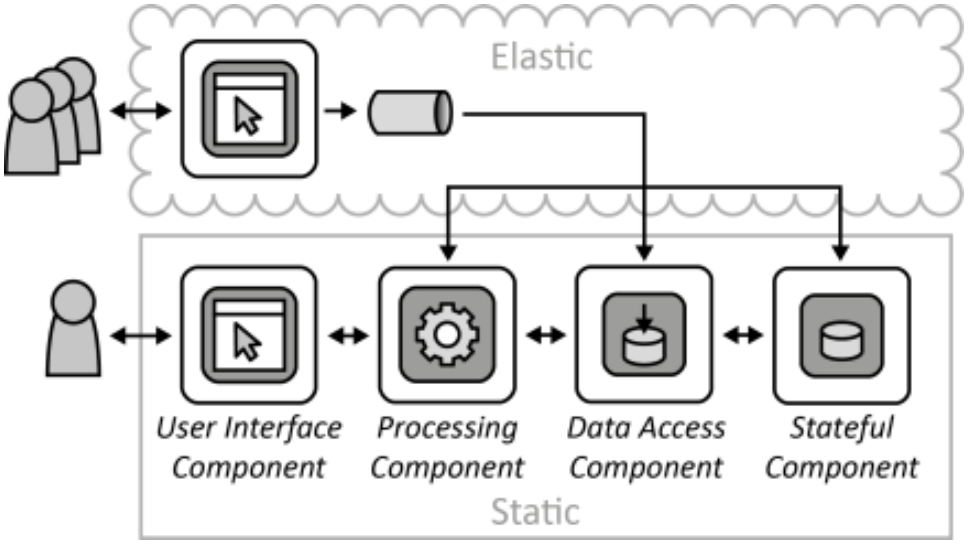
- Intent: Apps component instances and data handled by them are globally distributed to meet the access performance required by a global user group



- Solution: Content replicas are established in different physical locations of one or multiple clouds. During distribution of replicas, the topology of distribution networks is considered to ensure locality for all users. Replicas are updated from a central location

Hybrid User Interface

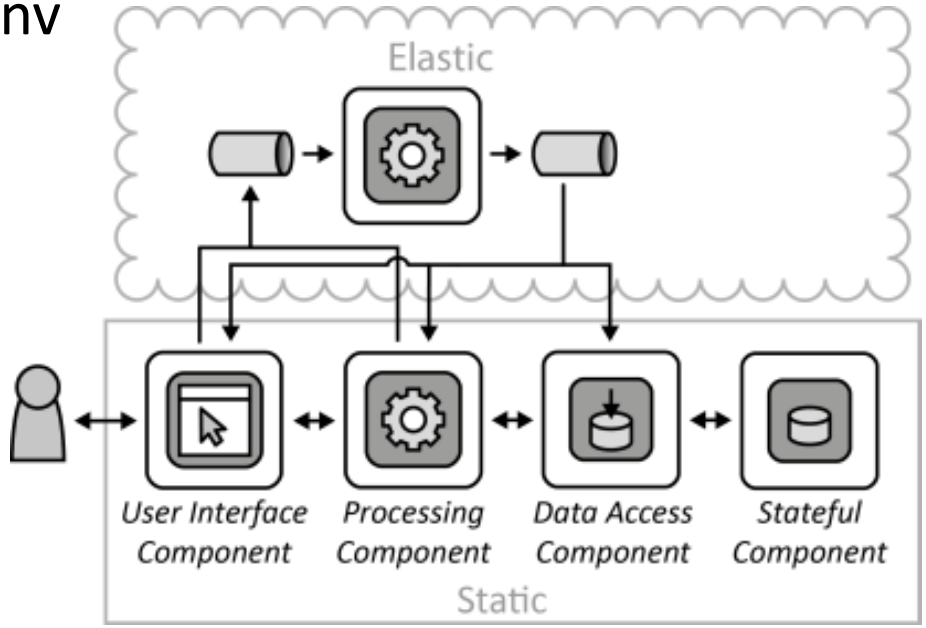
- Intent: Varying workload from a user group interacting async w/ an app is handled in an elastic env while the remainder of an app resides in a static env



- Solution: The UI Component generating varying workload is hosted in an elastic cloud env. Other app components are hosted in a static env. The UI is integrated w/ the remainder of app in a decoupled fashion using msg (loose coupling)

Hybrid Processing

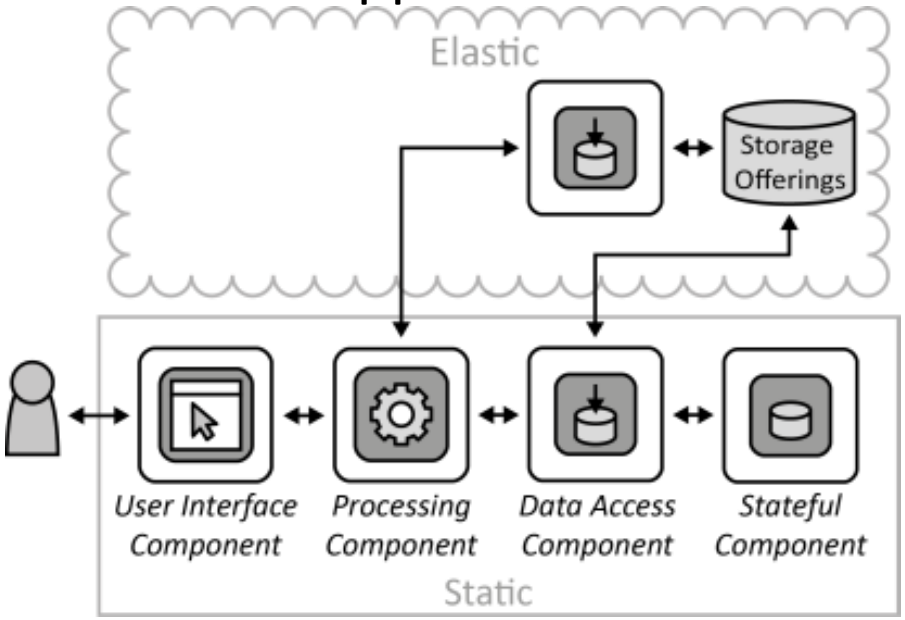
- Intent: Processing functionality that experiences varying workload is hosted in an elastic cloud while the remainder of app resides in a static env



- Solution: The Processing Components experiencing varying workloads are provisioned in an elastic cloud. Loose Coupling is ensured by exchanging info b/w the hosting envs async via msgs

Hybrid Data

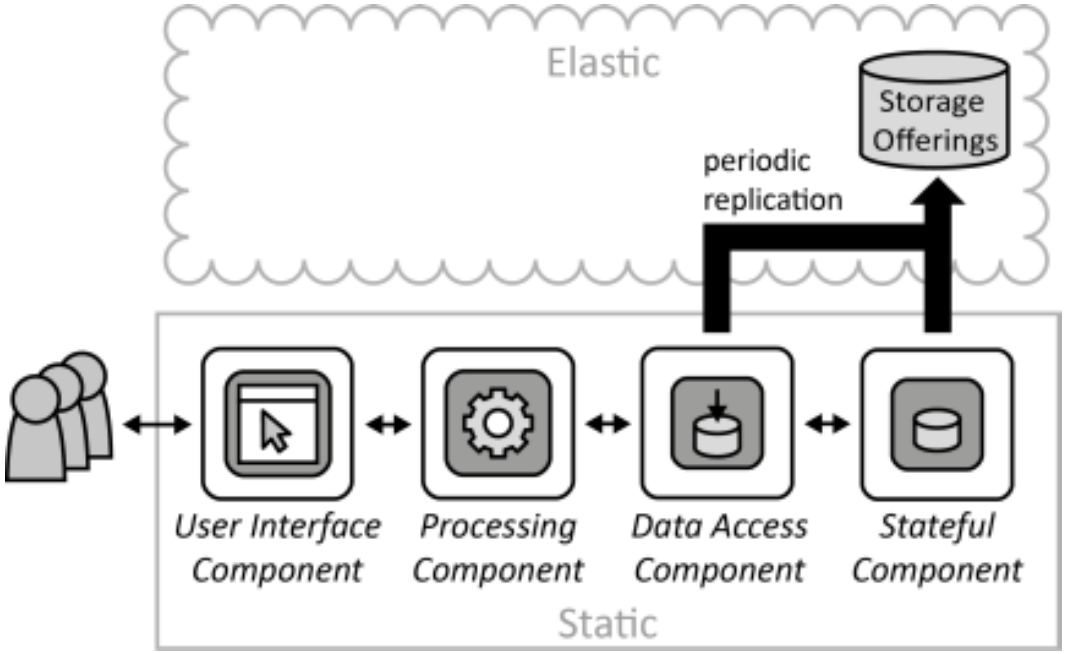
- Intent: Data of varying size is hosted in an elastic cloud while the remainder of an app resides in a static env



- Solution: Data whose varying size makes it unsuitable for hosting in a static env is handled by Storage Offerings in an elastic cloud. Data is accessed by Data Access Components that are either hosted in the static env or in the elastic env

Hybrid Backup

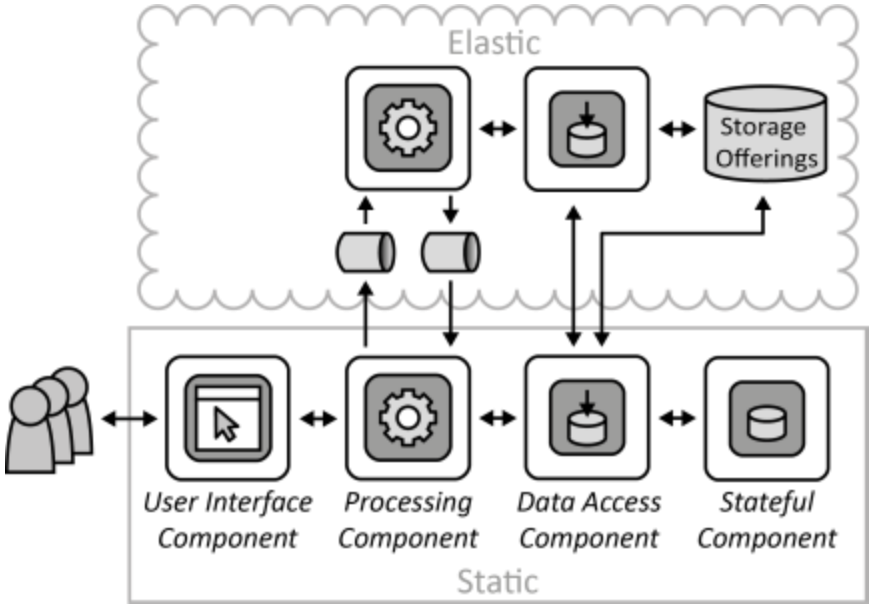
- Intent: Data is periodically extracted from an app to be archived in an elastic cloud for disaster recovery purposes



- Solution: A Distributed Application is hosted in a local static env of the company. Data handled by Stateful Components is periodically extracted and replicated to a cloud storage offering

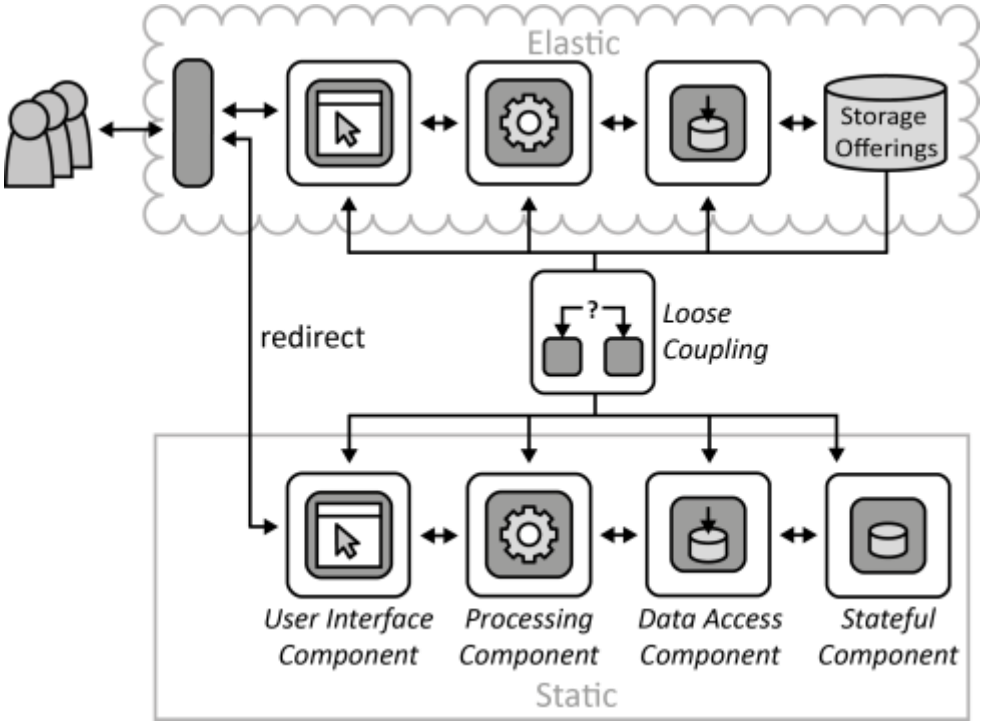
Hybrid Backend

- Intent: Backend functionality comprised of data intensive processing and data storage is experiencing varying workloads and is hosted in an elastic cloud while the rest of an app is hosted in a static data center

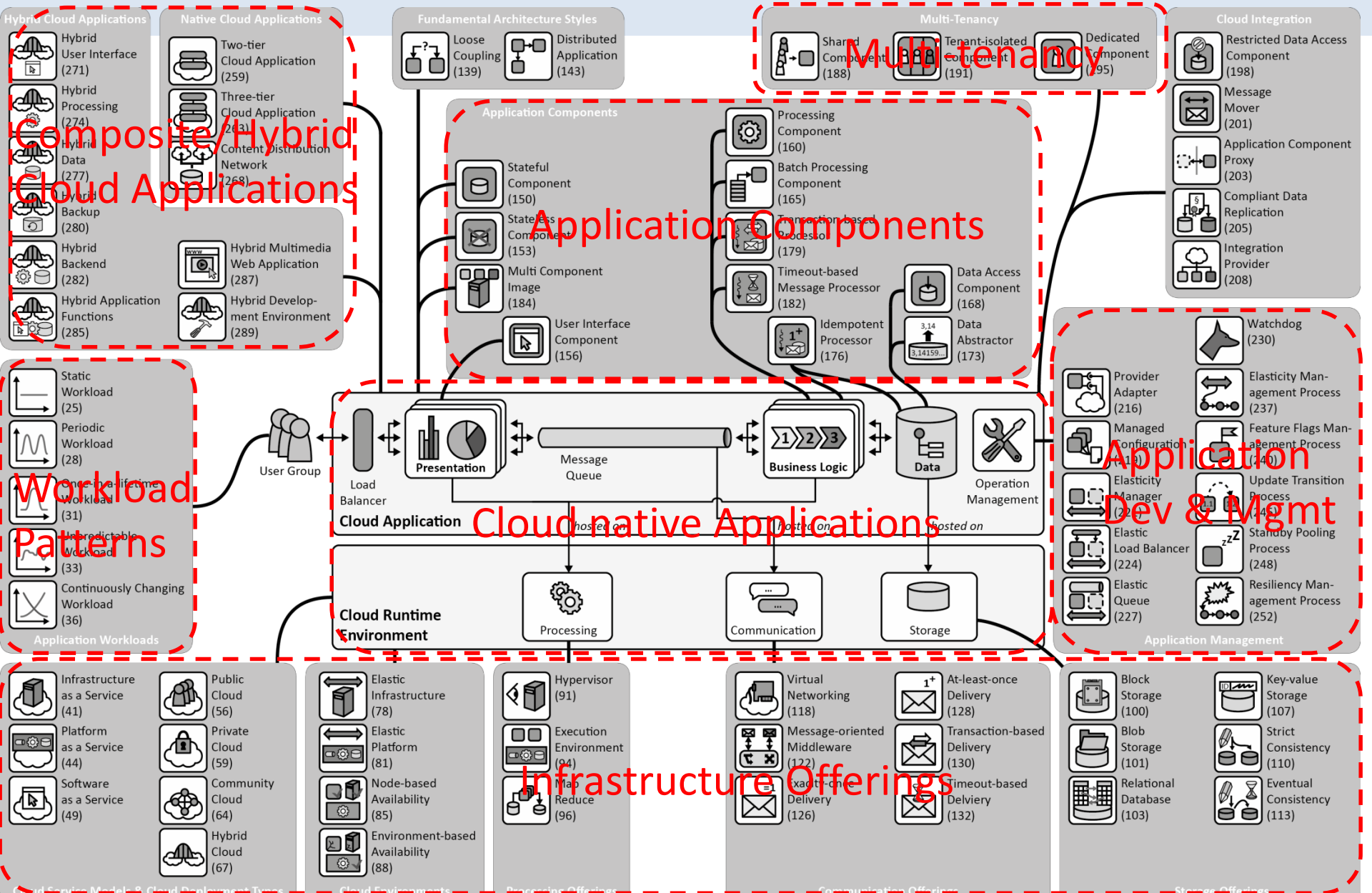


Hybrid App Functions

- Intent: Some app functionality provided by UIs, processing, and data handling is experiencing varying workload and is hosted in an elastic cloud while other app functionality of the same type is hosted in a static env



http://www.cloudcomputingpatterns.org



References

- http://www.cloudcomputingpatterns.org/
- Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter, *Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications*. Springer; 2014
  - http://www.springer.com/978-3-7091-1567-1
- The coffee shop example is adapted from
  - https://indico.scc.kit.edu/indico/event/26/session/1/contribution/12/material/slides/0.pdf
  - http://www.sei.cmu.edu/library/assets/presentations/retter-saturn2013.pdf