PaaS Design and Example PaaS Platforms

PaaS

- PaaS focuses on application developers
  - Allow developers to easily and rapidly develop and deploy the software, and continuous monitor and adapt its execution
- Encapsulate issues developers should not be bothered with
  - OS/Network/VM configurations, security patches, …
- Reduce the effort for developing cloud software
  - Via integration of apps and services
- Ease the rapid deployment of software

General PaaS Capabilities

- Development support
  - Support the development of "cloud native" applications
    - Which can take full advantage of the elastic cloud infrastructure
  - Support for porting existing applications
  - Provide development/testing tools
    - E.g., code repository/version control, test data generation and management, …
  - Support integration technologies
    - Service/App IO definitions
    - Service composition and App dependency definitions
    - Service/App management
      - Repository for storage, modification, search, …
      - Access control for the services and apps

General PaaS Capabilities

- Services and Apps
  - Various data stores
    - SQL and NoSQL of different venders
  - General services
    - E.g., business analytics services, rules engines, event processing services, mobile back-end services, …
    - Security services: e.g., user management services
  - Special functionalities
    - E.g., map service, speech service, image processing services, …
  - PaaS should manage these services and apps
    - Should not require the developer to install them
    - The execution should be automatically managed by PaaS (e.g., replication, configuration, monitoring, etc.)

General PaaS Capabilities

- Rapid deployment mechanism
  - Developer simply "push" the application
  - PaaS dynamically and automatically allocate resources
- Operation time support
  - Security tools
    - Firewall, endpoint management, secure commu. protocols, …
    - authentication and authorization, encryption, integrity checking, …
  - User management tools
  - Application monitoring and analytics services
    - E.g., logging, log analysis, monitoring and event handling
  - Execution adaptation tools
    - Auto-scaling, load balancing, …

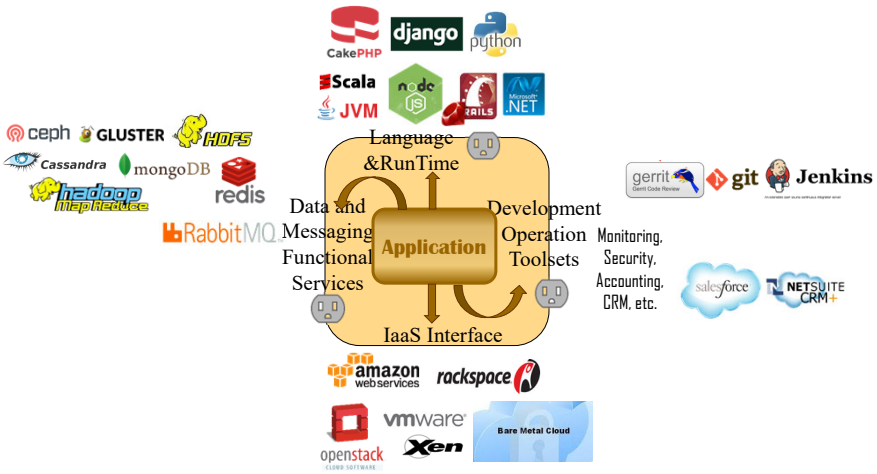
General PaaS Capabilities

- Simplicity and flexibility principle
  - Keep most parameters by defaults, but allow the developer to set the parameters if desired
    - E.g., resource demand configuration parameters
    - E.g., PaaS offers auto-scaling, users can also increase or decrease the number of running instances of an application via dashboard
  - Allow customers to easily plug in their own tools
    - New language, run time environment
    - New load balancing tools
    - External services and apps
    - …

Open PaaS Model

- Open PaaS requirements
  - Support multi-language and their run-time environments
  - Allow developer to integrate various services/apps from different sources
  - The PaaS can be deployed to multiple clouds/IaaSs
  - PaaS itself is open source
  - Allow customers to easily plug in their own tools
    - New language, run time environment
    - New load balancing tools
    - External services and apps
    - …

OpenPaaS Model

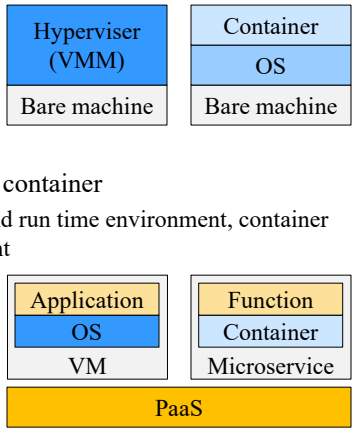


Microservice Concepts

- Some general concepts
  - Serverless function
    - Users focus on one function, not the overall system
    - PaaS supports the deployment of the function
  - Microservice architecture
    - Monolithic architecture: Generally has layered design, and the application has only a single unit to be deployed
      - May have concurrent units (like threads), but not externally visible
      - Component based and object-oriented are all in this category
    - Microservice architecture: Application is divided into small independent units that interact with each other
      - Development can be more independent
      - Easy to manage in the cloud environment, resources for each unit can be managed independently

Microservice Concepts

- Some general concepts
  - Virtual machine and container
    - Container has to run on OS
    - Container is very light weighted
  - Microservice and container
    - Each microservice is deployed in a container
      - Execution of a service needs OS and run time environment, container provides these in a very light weight
    - Some PaaS support microservice architecture (MA)
      - User can choose VM or MA



Microservice Support

- Kubernetes
  - An orchestration platform on top of Docker for managing multiple contains in one or more applications
- Pods and clusters
  - Pod
    - May contain one container or multiple interacting containers
    - A deployment unit, can be hosted by one or more hosts
      - Default is one, can use "nodeSelector" to specify the user assignments
      - Replicated on multiple nodes (by Kubernetes)
    - Kubernetes may scale-out pods dynamically for performance
    - Container interaction in a pod: user coded
      - Shared volume (file system)
      - System V IPC: shared memory

Microservice Support

- Kubernetes pods and clusters
  - Cluster
    - Can be viewed as a physical and a logical unit for an application
    - Can have multiple interacting pods in the application
    - Multiple hosts work for the same application
      - Worker node: pods are allocated on them
      - Master: manage the status of the worker nodes, pod allocation, load balancing, monitoring, …
    - Can use KubeCtl to manage the cluster
    - Communication between pods
      - Using IP and port number (too primitive)
      - Use http to communicate with a backend pod
        - Assign labels to pods, select pods as backend pods using the label
        - E.g., "http://backserv:8080" (backserv is the service name)
      - Deploy pods as services => same internal and external commu

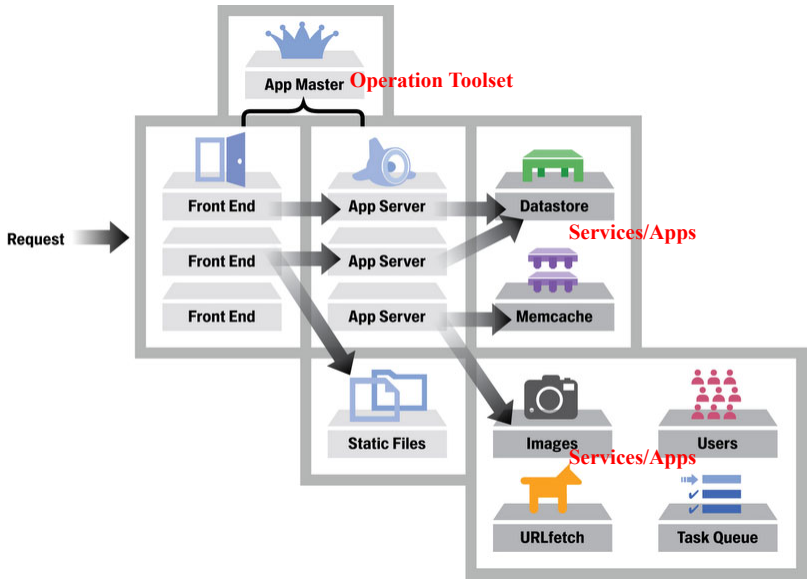
Microservice Support

- Docker Swarm versus Kubernetes
  - Installation and setup
    - Kubernetes: manual installation can differ for each operating system, but additional tools are more widely supported
    - Swarm: simple installation, consistent across operating systems.
  - Both offers autoscaling
    - Kube is automated, Swarm needs activation
  - Both offer availability by default replication
    - Swarm: automatic load balancing, Kube: need external balancer
  - Other
    - Kubernetes supports easy service discovery via a DNS server, allow access to containers via IP address or HTTP route
    - Swarm access is via deployed services

Microservice Support

- Docker Compose
  - A tool for deployment of containers
  - Communication between containers
    - Can create a network object in Compose, each container can attach to the same network => communicate via http requests
      - Docker Compose offers a default bridge network for all containers deployed together in the compose file
    - External to Compose
      - Use IP and port OR use shared volume OR use shared memory
  - Docker Compose versus Swarm
    - Compose supports container deployment, but only on a single node
    - Swarm is not specific for deployment, manages multiple hosts

Google App Engine



Google App Engine

- Development
  - Support multiple languages Java, Python, PHP
  - Use eclipse, and support some simple testing
    - Check whether the service (e.g., data store) usages are correct
  - Support time-based task activation
    - GAE cron, like Unix cron, activate tasks at a scheduled time
    - Task queue API: queue tasks for controlled activation
  - Workflow support
    - GAE pipeline API: can be nested and dynamic
    - Fantasm: state machine based and external to code (API is at the code level)

Google App Engine

- Deployment
  - Build a project and deploy the application via App Master
    - Can only deploy on Google IaaS
    - User first creates a cloud platform project and then deploy the application on the project platform
      - Need to specify the region or the zone in a region for the platform
      - Can specify the resources for the platform, if desired
- Monitoring and management after deployment
  - Monitoring system provides "usage report"
  - User can manage the resources according to the usage
    - Control traffic split to multiple instances of an application, increase/decrease resources, control auto-scaling behavior (tradeoff between better performance or minimal resources), …

Google App Engine

- Microservice support
  - Run each microservice just like a regular GAE application
    - Fully isolated via project boundaries
  - Run multiple microservices in one GAE application
    - Each service is deployed as a service (former module)
    - Have isolated resources, but share data stores, Memcache, and TaskQueue

Amazon AWS Cloud

Amazon AWS Cloud

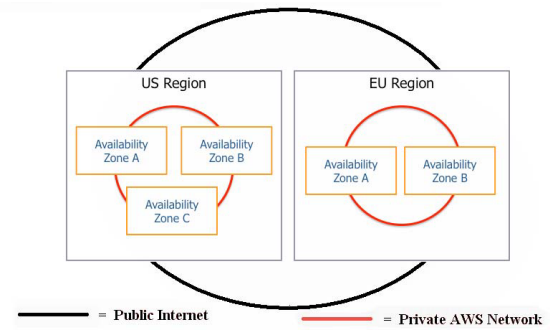
- Compute Service
  - EC2
    - Users request resources (by specifying CPU, memory, storage, and networking) and build a VM
    - Support auto-scaling, etc.
  - Amazon Elastic Beanstalk
    - Just upload application files, EBS creates the instances for you
  - Lambda
    - Serverless compute service, triggered by specified events
  - AMR
    - AWS's built-in map reduce solution

AWS Compute

- EC2 instance
  - Users can create a login and then create an EC2 instance
  - An EC2 instance can be N VMs, user specifies
    - Hardware configurations: cores, memory, disk
    - Image: to run on each VM
    - Key pair: credentials for accessing VM
    - Region: Geographical location
      - Associated with price, laws, network locality
    - Availability Zone = Subdivision of region
  - Additional
    - Can specify the auto-scaling option: scale up/out automatically
      - What are scale up and scale out
    - Can attach other services (e.g., storage, messaging, security, etc.)

AWS Compute

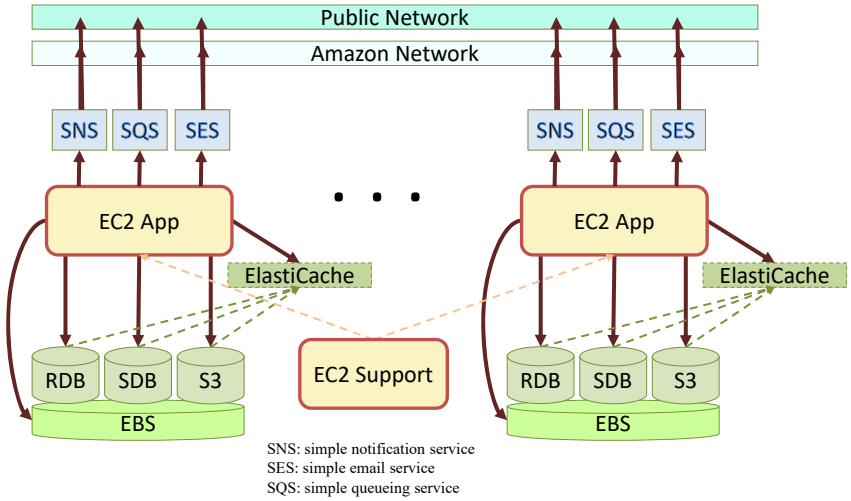
- EC2 regions
  - Cross region: public network
  - Within each region: can have private network



http://aws.amazon.com/ec2/

AWS Compute

- General architecture to support EC2 applications



AWS Compute

- AWS Lambda
  - An event based compute service
    - User specifies the events and code the corresponding actions
    - User deploys the event handling code as the Lambda functions
    - Lambda provides all the deploying environment, run time event delivery and handler activation
    - Event can be http requests, data store changes, some monitored events (failures, threshold reached, …), etc.
  - No need to allocate resources, etc.
  - But is not standalone, needs to have other services to form a complete application
    - Need the other services to trigger the lambda functions

AWS Compute

- AWS Lambda
  - An example use case
    - Front-end: Build a web site on S3 for Image sharing
      - Create an S3 bucket, specify its property as "static web hosting"
      - Create web pages (HTML, JavaScript) and load them to the bucket
        - Should specify the index document, e.g., index.html
      - Can also specify the "bucket policy" to control the page accesses
    - Lambda function for image standardization and storage
      - Event for Lambda: entry into an S3 object
      - The function resizes the image and stores in in Dynamo DB
    - Lambda function for image processing or integrated analytics
      - Event for Lambda: insertion in Dynamo DB
  - In this application, user do need to allocate DB resources

AWS Development

- Development environment
  - Simple workflow service (SWF)
    - Activity worker
      - Each entity in the workflow is an activity, each activity worker executes the software/service for the activity
    - Decider
      - Workflow controller, activate the activity workers at the proper situation with proper logical control (such as if and loop)
      - Decider is also to be coded
      - More flexible control, more user involvement
    - AWS Flow
      - Manage the communications between activities
      - Deliver the output of one activity to another and activate activities
      - No longer supported
    - Use Step Function instead

AWS Development

- Development environment
  - AWS Workflow Studio
    - Generate a workflow: a state machine of lambda functions
    - States of each step in the state machine
      - Pass: pass the output to the input of next node
      - Choice: a choice rule to determine the flow (rule can involve the data in transition: output to input)
      - Parallel: parallel execution of subsequent branches
        - The data flow involves key-value pairs that can be delivered based on rules (e.g., some keys to a certain branch)
      - Map: deliver the data to all subsequent nodes and execute them all
      - Wait: specify the delay of the flow in seconds
      - Succeed, Fail: both terminates the execution, with different status

AWS Storage and DB

- S3 (simple storage service)
  - Bucket and objects
    - Under bucket, user can create folders and/or objects
      - Folders are just an illusion, do not actually exist
      - Objects are files with metadata, access by key like "a/b/c"
  - Can be viewed as a key-value store, but is not a DB
    - The value filed is always a file
  - Each object can be accessed via http externally
- EBS (elastic block storage)
  - Raw block storage, can be mounted to an EC2 instance
    - Can only be accessed by an EC2 instance
    - Can create a file system or a database on top of it, or just use it raw

AWS Storage and DB

- EFS (elastic file system)
  - A real file system like Unix FS
  - Can be mounted to an EC2 instance
- Elasticache
  - In memory storage, not persistent, uses Redis
- RDS (relational database service)
- DynamoDB
  - Can create a table that is accessible via http

AWS Storage and DB

- All AWS storage solutions
  - All are automatically replicated in the same zone
    - User can specify cross zone replication and number of replicas
  - S3, Easticache, RDS are instances in an availability zone
    - During creation, need to specify the zone
    - Can be access externally via http
    - Need to specifically build the private network service (VPS) to enable EC2 to storage connection
  - EBS and EFS are only attached to an EC2 instance
    - The zone is the same as the EC2 instance
  - DynamoDB table creation
    - Does not require zone specification, automatically has cross zone replication

AWS Administration

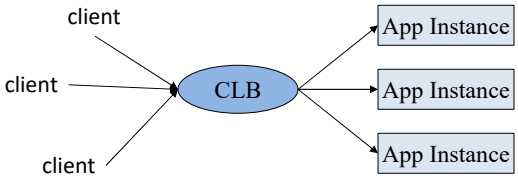
- CloudWatch service
  - Monitoring the cloud instances, computing, storage, etc.
    - Select which entities to watch, which metrics for that entity to watch, and at what frequency
  - Set alarms on any of the selected metrics
    - Can simply receive notifications or take other automated actions when the metric crosses the specified threshold
  - View statistics and graphs for any selection
  - Monitoring configurations can be adapted dynamically
    - Reported latency of 15 minutes
  - Customer can specify their own metrics and set to run in the CloudWatch framework (called custom metrics)

AWS Administration

- Elastic Load Balancing (ELB)
  - Add ELB in front of an application to receive its client requests and route them to multiple application instances
  - Also monitor the healthiness of each node
    - Do not route to nodes that are identified to be unhealthy
  - https://aws.amazon.com/elasticloadbalancing/features/
  - AutoScaling (scale in/out)
    - Is not an AWS service, only associated with an EC2 instance
    - Triggered by the policy in CloudWatch

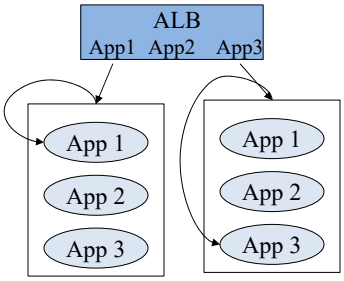
AWS Administration

- ELB
  - Classic load balancer
    - Router (CLB) receives all client requests
      - Use one IP and one port (same IP for the same application)
    - Router forwards requests to the application instances
      - Round Robin only



AWS Administration

- Management after deployment
  - Application load balancer
    - Router can handle multiple applications
      - Multiple http's captured by the router and router routes according to http's (also support TCP based routing)
      - Routing can be based on the group
        - » Consider all applications in a group
        - » May generate new internal load (a service calls another)
      - Or based on individual applications
    - Balancing policies
      - Round Robin
      - LOR
        - » ALB forward load to the node with the least outstanding requests (LOR)



AWS Security

- Identity and access management (IAM)
  - Manage users and their accesses to the resources
    - E.g., S3 requires to have an IAM instance to protect its accesses to each bucket or each object
  - Can create users and groups, and roles
    - Roles are permissions to resources
- Cognito: support external identity management
  - E.g., can use google, facebook, etc. logins
- CloudTrail
  - Logs users/accounts that accessed a monitored resource
- GuardDuty: a threat detection service
  - Work with CloudWatch for setting up alerts/reactions

AWS Deployment

- Deployment
  - A very important feature in PaaS
- Deployment with various flexibility
  - Previously
    - All the instances need to be created independently
    - These can be done in an integrated way via CloudFormation
  - CloudFormation
    - Can specify all the required resources in one template and deploy them in one step
    - But need LAMP to help with application deployment
      - Specific parameter settings in various applications are not supported in CloudFormation (e.g., password setting for DB)

AWS Deployment

- Deployment with various flexibility
  - Opsworks
    - Automatically deploy applications with pre-configuration
    - Use Chef to help with the resource configurations
    - Use Puppet to help with the application software deployment
      - User still need to make dependency specifications, etc.
    - Best usage scenario
      - Continuous modification of the application software
      - Follow the paradigm of SaaS and microservice architecture
  - Elastic Beanstalk
    - Automated deployment, user just provide the application containers with a few configuration Parameters
    - All resources are decided by Beanstalk
    - Most suitable for three-tier applications: a front-end, an app service, and a DB service

Microsoft Azure

- Azure Compute
- Azure Storage
  - Table, Queue, Blob
- Azure AppFabric
  - Access Control, Caching, Service Bus
  - Windows Azure Fabric Controller
- Azure Virtual Network
  - Azure Connect and Azure Traffic Manager

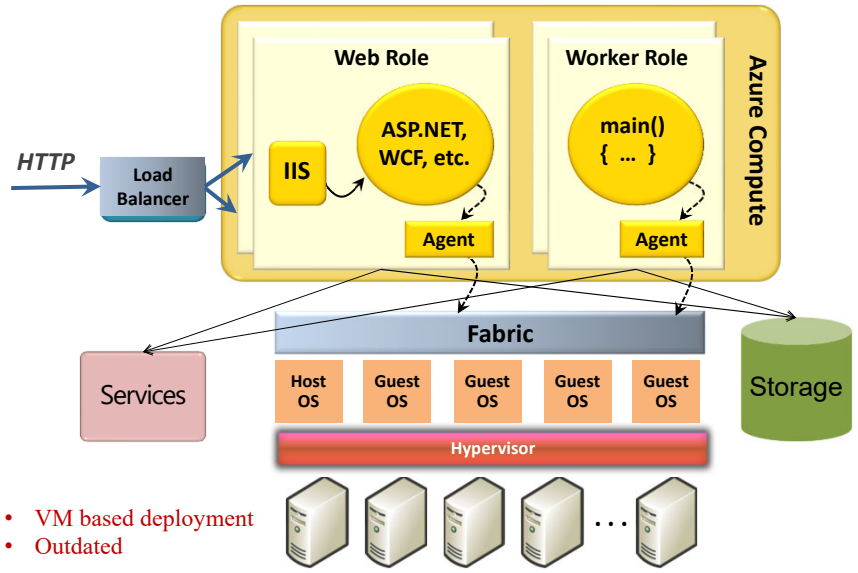
Azure Compute

- A compute unit is a "service", it has
  - Definition information
  - Configuration information
    - Number of instances, fault domains, etc
  - Roles
    - Multiple levels of encapsulation for compute module development and deployment
    - Web Role: IIS or .NET
    - Worker Role: Generally the customer code
    - VM Role: customer code on customer OS
      - IaaS model of compute
  - A compute unit can access Azure storage and services

Azure AppFabric

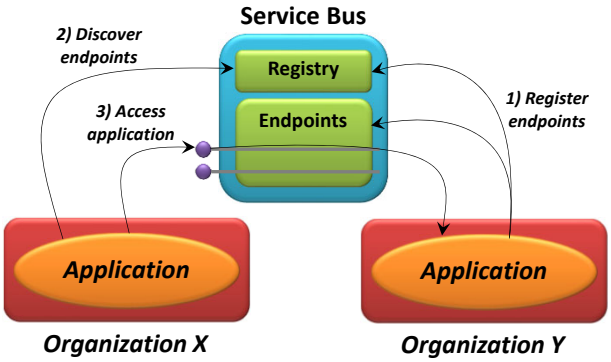
- Azure AppFabric
  - Deploy user application and run it in the cloud
  - AppFabric service bus
    - Provides a service registry (like UDDI)
    - Provides mediation between clients and services, a client request can be forwarded immediately or queued
  - AppFabric access control
    - User setup: Choose the identity provider ⇒ provide security rules ⇒ Configure the application to use the rule set
- Windows Azure Fabric Controller
  - Decides where a new application should run
  - Monitoring all running applications

Azure Compute and AppFabric



- VM based deployment
- Outdated

Azure Service Bus

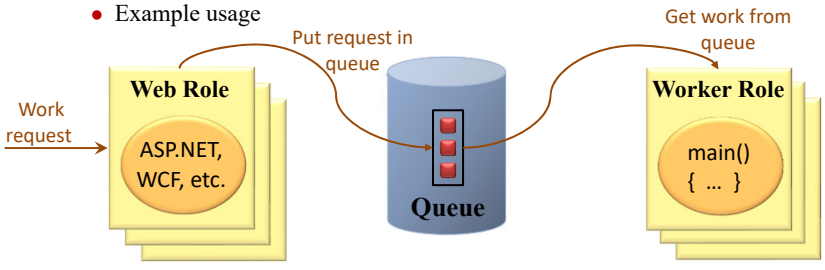


Azure Service Fabric

- Service Fabric replaces AppFabric
  - Still similar, but support microservice architecture
  - Orchestrator for resource management
    - User deploys the microservices in an application
    - Specify the resource demands and #instances of each microservice
    - Orchestrator allocates resources for the microservices
  - Continuously monitoring the health of the application
  - Dynamically scale in/out
- IaaS independent
  - Can be deployed on any IaaS cloud

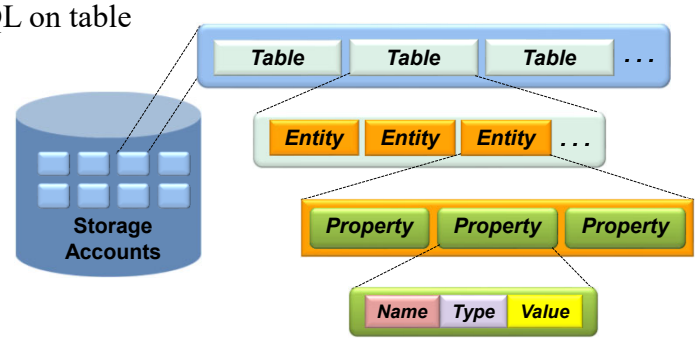
Azure Data Store

- Windows Azure Storage
  - Blob
    - Similar to conventional file system, stores files and metadata
  - Table: Provide structured storage
  - Queue
    - Provide reliable storage and delivery of messages for applications
      - Example usage



Azure Data Store

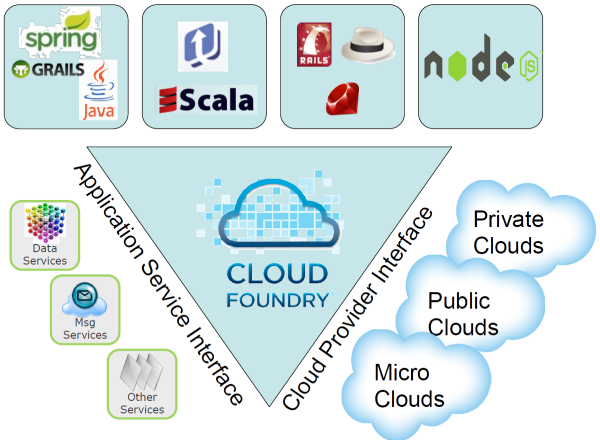
- Table structure
  - Entity is a data unit, it has to have three properties: partition key (multiple rows form a partition), row key, timestamp
    - No column concept, because properties of entities may be different
    - Not space efficient, but can support parallel data processing
  - No SQL on table



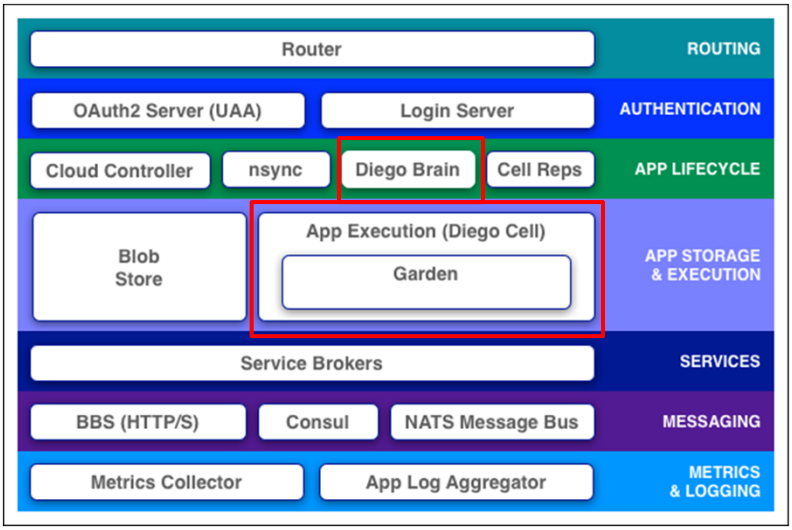
Development Tools in Azure

- Development tools
  - Visual studio, visual studio team, Jenkins, Eclipse
  - Azure DevTest Labs
  - Service bus and app APIs
- Deployment and operation
  - Service Fabric
  - Application Insights: for problem detection/diagnosis
  - HockeyApp
    - Mobile app deployment
    - Continuous monitoring, feedback collection, …

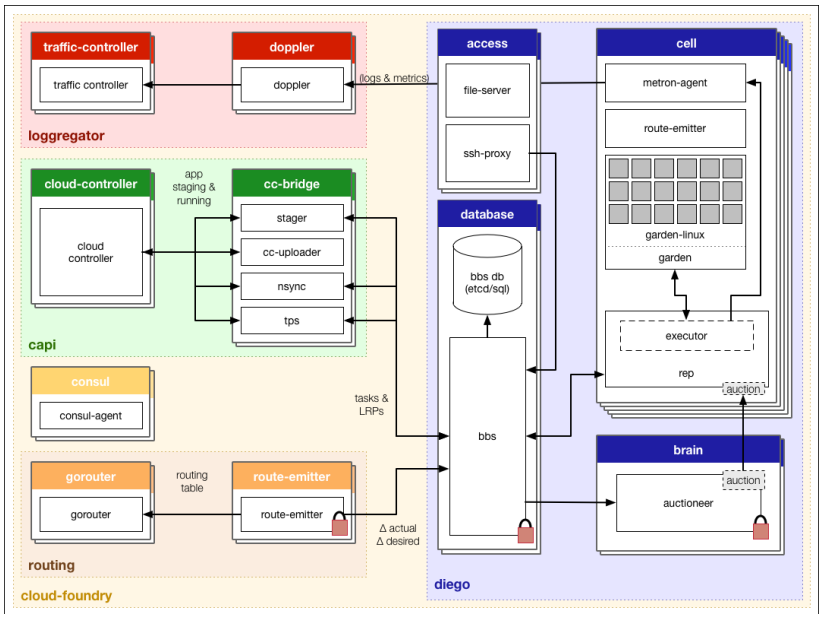
Cloud Foundry: An Open PaaS



Cloud Foundry Components



Cloud Foundry and Diego Architecture



Cloud Foundry and Diego Architecture

- Diego
  - The container management system for Cloud Foundry
    - An application runs in CF within a container
    - Warden: old CF's own container
    - Docker: almost the de facto container
    - Garden: new CF's container, can fetch and run a Docker image
  - Diego Brain
    - Distribute applications to Diego Cells
    - Have an "Auctioneer" to make resource allocation decisions via an auction scheme (developed by CF)
    - Communicates to Cells regarding auction activities via Cell Reps

Cloud Foundry and Diego Architecture

- Diego
  - Diego Bulletin Board System (BBS)
    - Maintains the state of the Diego cluster, including all applications and its instances
      - #instances for each application
      - The locations (VMs) and status of the instances
    - Provide a messaging mechanism for the Diego components
      - Monitor missed messages, resending them if necessary

Cloud Foundry and Diego Architecture

- Diego
  - Diego Cell
    - Cell Rep
      - Represents a Cell in Diego Auctions for task allocation
      - Mediates all communication between the Cell and the BBS
      - Maintains the presence of the Cell in the BBS
      - Ensures that the application has its desired #instances is maintained
      - Has an Executor to: (1) execute applications and (2) stream stdout and stderr to the Metron agent running on the Cell
    - Metron Agent:
      - Forward logs, errors, and other monitoring metrics to Loggregator
    - Route emitter
      - Monitor potential route changes due to configuration changes or failures and update the routing table
      - Emit the routing table, when necessary, to the BBS

Cloud Foundry Components

- Router
  - Route incoming traffic to the appropriate component
    - A new application "push" is routed to Cloud Controller
    - A request related to a hosted application is sent to its VM (then down to cell and container)
  - Periodically queries BBS (Diego bulletin board) about
    - Which cells and containers each application currently runs on
    - The IP address of each cell's VM and the port number for each containers of each cell
  - Modify routing tables
    - From the BBS query result, if there are changes
  - Route-emitters of individual cells may communicate with router about their updates

Cloud Foundry Components

- Cloud Controller (CC)
  - User push an application A to Cloud Foundry
    $\Rightarrow$ Router forward it to CC
  - CC then manages the deployment of A
    - CC directs the Diego Brain through the CC-Bridge components to coordinate individual Diego cells to stage and run applications
  - CC also manages orgs, spaces, user roles, services of A

PaaS Support for Access Control and Multi-Tenancy

Conventional Access Control Models

- Access control matrix
  - A: $S \times O \rightarrow R$
    - A: access control matrix
    - S: subjects, users or processes
    - O: objects, resources such as files, devices, messages, etc.
      - An entity could be a subject and an object
    - R: rights (or actions on O)
  - Example:

	object 1	object 2	object 3
subject 1	R W	R	R W E
subject 2	R	R W	R

Conventional Access Control Models

- Alternatives to ACM
  - Generally ACM is very sparse
  - Access control list (ACL)
    - Specify the access policies for each object
  - Capabilities list
    - Specify the access rights each subject has
- ACM does not cover
  - Time constraints
    - E.g., only allowed to access at night
  - Historical based constraints
    - E.g., A user can only write to a buffer 3 times (can be for DDoS, defense against denial of service or other protections)

Conventional Access Control Models

- Optimization
  - In a real system, may have too many subjects and objects
  - Unix
    - Classify subjects into: owner, group, world
    - Use ACL for each object, but in terms of owner, group, world
    - Very easy, but has some limitations

Conventional Access Control Models

- Discretionary access control (DAC)
  - Owner determines access rights
    - E.g., Unix ACL
  - Mandatory access control (MAC)
    - System enforces system-wide rules for access control
    - Security officer of a domain defines the AC policies
    - Still can use ACM, ACL for policy specification, but too high an overhead, generally use other models for easy management
      - Multi-level access control
      - Lattice model

Conventional Access Control Models

- Bell-Lapadula model
  - A formal model of military security concept
    - Information (objects) has different sensitivity levels or classifications (denoted as L(o))
      - E.g., top secret > secret > confidential > unclassified
    - Subjects have different clearance levels (denoted as L(s))
    - "No read up" policy
      - Rule: s can read o if and only if $L(s) \geq L(o)$
    - "No write down" policy
      - Rule: s can write o if and only if $L(s) \leq L(o)$
      - Write down will cause information to flow down
    - Information can only flow up, not down
- Also called multi-level access control

Conventional Access Control Models

- Role based access control
  - Can be mandatory or discretionary, but mostly used as mandatory
  - Match the semantic of real world scenarios
    - Alice works in sales → sales role, access to product catalogs
    - Alice switches to personnel → access to employee records
    - Easy switch, just change user to role assignment
    - Bob takes over the sales job → sales role, access to prod. catalogs
    - Bob is also in charge of personnel → access to employee records
  - Benefits
    - Smaller numbers of relations, if r is much smaller than m
    - Roughly from O(mn) to O(mr+nr)
      - m: number of users; n: number of resources; r: number of roles
      - mr user to role mappings, nr role to permission mappings

Conventional Access Control Models

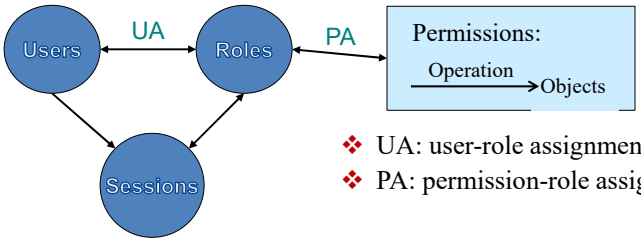
- Role based access control benefits

TABLE 1: ESTIMATED TIME (IN MINUTES) REQUIRED FOR ACCESS ADMINISTRATIVE TASKS

TASK	RBAC	NON-RBAC	DIFFERENCE
Assign existing privileges to new users	6.14	11.39	5.25
Change existing users' privileges	9.29	10.24	0.95
Establish new privileges for existing users	8.86	9.26	0.40
Termination of privileges	0.81	1.32	0.51

Conventional Access Control Models

- Role based access control
  - RBAC-0: Core RBAC
    - A user can have multiple roles
    - A role can be assigned to multiple users
    - A role can have multiple permissions
    - A user can have multiple sessions
    - For each session, the user can assume a subset of the roles



- UA: user-role assignment
- PA: permission-role assignment

Conventional Access Control Models

- Role based access control
  - RBAC-1 = RBAC-0 + Role hierarchy
    - Role hierarchy also matches with real world role semantics
      - Director to manager to employee
    - Senior roles acquire the permissions of their juniors
    - Further reduces the number of permission assignments

Conventional Access Control Models

- Role based access control
  - RBAC-2 = RBAC-0 + Constraints
    - Constraints can be added on roles
      - Use constraints to realize separation of duty (SoD)
    - Static separation of duty
      - Constraint: A user should not be assigned to conflicting roles
    - Dynamic separation of duty
      - Constraint: A user should not activate a session that has conflict roles
    - Enforce CoI through SoD
      - Define CoIs as conflicting roles
      - Less structured than CW mechanism
        - » No specification of CoI groups
        - » Need to map all CoIs to conflicting roles (pair-wise)

Conventional Access Control Models

- Role based access control
  - RBAC-2
    - Cardinality constraints
      - E.g., confine # of roles an individual can be assigned to
    - Prerequisite roles
      - A user cannot assume a role without previously being assigned another role
      - E.g., a person cannot assume the physician role without having been assigned the intern physician role
  - RBAC-3 = RBAC-0 + Role Hierarchies + Constraints

Azure

- User management
  - Support group definitions
- Role based access control
  - Can assign access rights of resources to roles
  - Can assign roles to users
  - Can use build-in roles or have custom-defined roles
  - But no role hierarchy

Azure

- Multi-tenancy
  - Client view
    - User sign in with organizational account (e.g., x@y.com)
    - User access a resource (application) with organization credential
    - Organization assigns roles to each user (access rights)
  - Mechanism
    - Azure active directory (AD): An organization can have one or more active directories which manages its user identities
    - Subscription: Subscription to Azure services (e.g., outlook, office, storage, service bus)
      - Each AD can have multiple subscriptions
    - Resource group and resources

Azure

- Multi-tenancy
  - Mechanism
    - Resource
      - A manageable item that is available through Azure
      - E.g., a virtual machine, storage account, web app, database, etc.
    - Resource group
      - Each subscription can have one or more resource groups
      - Each resource belongs to one resource group
      - Resource group cannot be nested
    - The groups are defined to facilitate easy management of the subscription to Azure services
      - Assign access rights in groups, assignment can be inherited
      - Addition/deletion of resources can be done in a logical group
      - Assignment can also be at the subscription level

Azure

- Multi-tenancy
  - Deployment
    - Deploy the service as regular services and configure its
      - Geo-location, #instances, auto-scaling, …
    - Developer has to develop the DB solution for multiple tenants
      - Azure supports multiple DB accounts for the shared DB
        - » DB can be partitioned
        - » If the user has the account key, user can read/write the DB partition for the account

Cloud Foundry

- User account and authentication (UAA)
- Isolation via orgs and spaces
  - CF can have multiple orgs, an org can have multiple spaces
  - Each application/service is deployed in a space
- Roles
  - Has a set of predefine roles, no user defined roles
    - Access rights can be defined in the scope of orgs/spaces
    - Read/write in an org/space, assign user-role in an org, …
  - A user can be assigned to a set of roles
- Access rights
  - Service access control are defined by orgs/spaces
  - No specific support for data access control

Cloud Foundry

- Multi-tenancy
  - Orgs is the structure to support multi-tenancy
    - A user belongs to an org (tenant)
    - An application is deployed to a space or org
    - How to make services accessible by multiple orgs?
      - Register its service broker with cloud controller
      - Publish the service plan to multiple orgs (or all orgs)
    - How to maintain DB for multiple tenants
      - Not well supported
      - Developer may have to manage the DB to achieve multi-tenancy access control

In General

- Most of them
  - Claim to consider RBAC, but actually are just assign individual access rights to a certain "space"
    - Microsoft considers actual RBAC
  - Do not consider cross-tenant accesses
    - Each "space" is a silo, not allowing crossing the silo boundary
  - Not flexible in terms of using different access control models
  - Do not consider the information flow problem

Readings

- Web resources
  - Practical Guide to Platform-as-a-Service Version 1.0", Cloud Standards Customer Council
  - https://cloud.google.com/appengine/docs/flexible/
  - https://aws.amazon.com/cn/documentation/
  - https://docs.microsoft.com/en-us/azure/
  - http://docs.cloudfoundry.org/