DECIDE Cloud Patterns Compendium

Introduction

This compendium contains a set of architectural patterns, from various sources, that are recommended by the DECIDE project for multi-cloud aware software applications. The set of patterns introduced in this compendium are meant to aid developers to design their applications in a way that it is multi-cloud aware and ensure that a set of non-functional requirements are always fulfilled when the application is running.

The main sources of architectural patterns that are used for the compilation of this compendium are:

- Cloud Computing Patterns: Fundamentals to Design, Build, and Manage Cloud Applications
- CloudPatterns.org – Cloud computing Design Patterns
- AWS Cloud Design Patterns
- Microsoft Azure Cloud Design Patterns

Each architectural pattern description contains the problem statement, the context and the solution. It also contains the relevant model if available. A link to the source of each pattern can be found at the end of the description.

Table of Contents

1 Dynamic Failure Detection and Recovery

When cloud-based IT resources fail, manual intervention may be unacceptably inefficient.

1.1 Problem

How can the notification and recovery of IT resource failure be automated?

1.2 Context

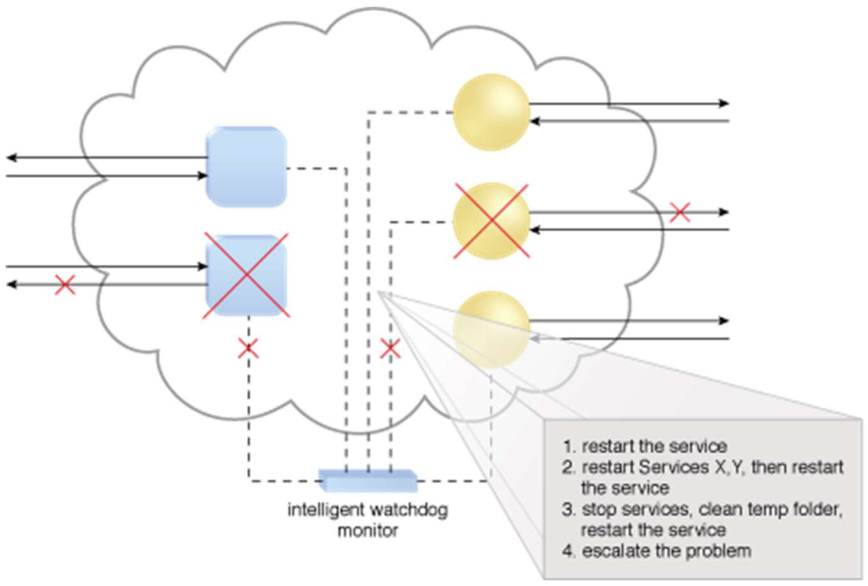
A watchdog system is established to monitor IT resource status and perform notifications and/or recovery attempts during failure conditions.

1.3 Solution

Different intelligent monitoring and recovery technologies can be used to establish the automation of failure detection and recovery tasks with a focus on watching, deciding upon, acting upon, reporting and escalating IT resource failure conditions.

Source: Dynamic Failure Detection and Recovery

1.4 Model



1. restart the service
2. restart Services X,Y, then restart the service
3. stop services, clean temp folder, restart the service
4. escalate the problem
intelligent watchdog monitor

2 Trusted Cloud Resource Pools

Cloud platform pool security needs to be achieved to meet cloud consumer compliance and regulatory security requirements. Verification of the platform assurance level is critical for regulated industries.

2.1 Problem

How can cloud-based resource pools be secured and become trusted?

2.2 Context

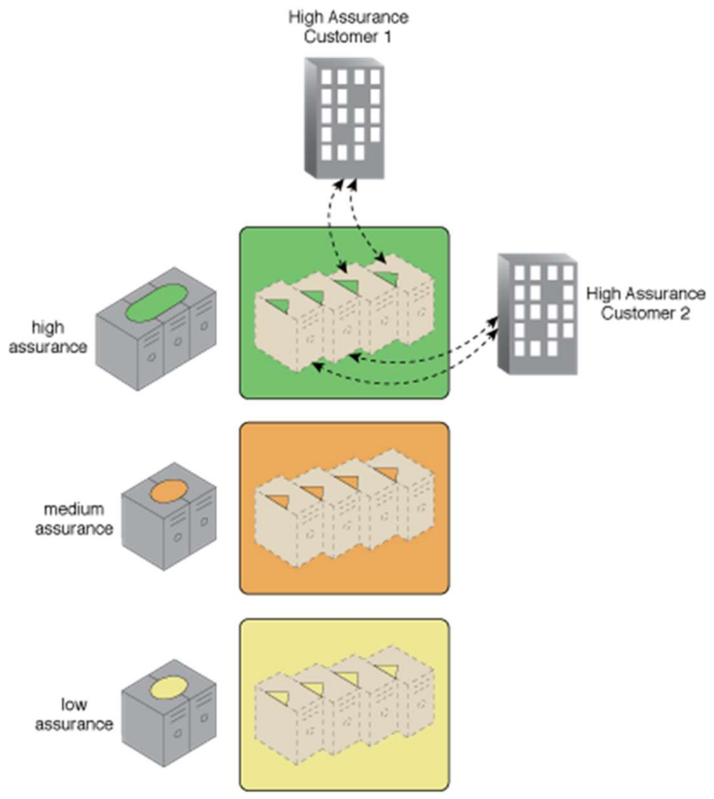
Trusted resource pools made up of trusted geotagged computers are made available by the cloud provider, and can be verified by the consumer through direct monitoring or evidence through auditing.

2.3 Solution

Achieving security through the use of trusted platform modules (TPMs), validating digitally signed code, geotagging, and remote monitoring of the platform security status, cloud consumers can verify that they are using compute platforms that meet their security assurance requirements.

Source: Trusted Cloud Resource Pools

2.4 Model



3 Cloud VM Platform Encryption

VM backups, snapshots and live migration create files that encapsulate the entire VM. These files can then be copied or moved outside the application that the cloud consumer controls, making them vulnerable to attacks.

3.1 Problem

How can VM backups, snapshots, and live migration be secured?

3.2 Context

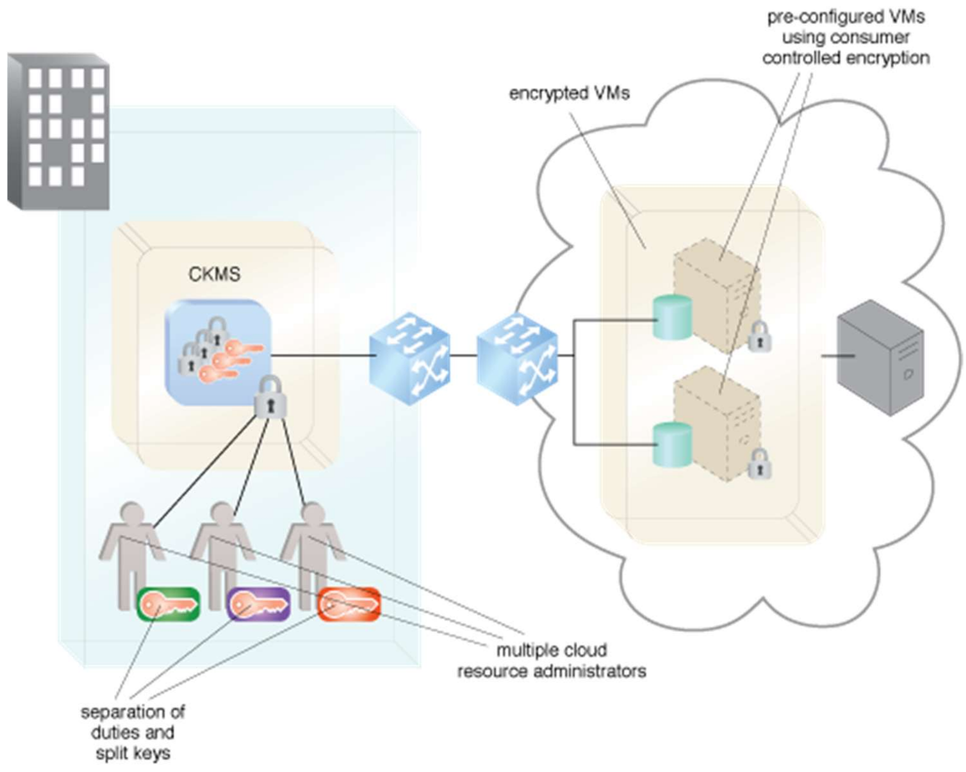
Encrypted containers are provided for use and storage of the various types of VM backups and replications.

3.3 Solution

A key manager is used to manage keys for encryption of the various types of VM storage that are pre-provisioned to receive backups and snapshots of consumer VMs or to receive replications and live migrations.

Source: Cloud VM Platform Encryption

3.4 Model


pre-configured VMs using consumer controlled encryption
encrypted VMs
CKMS
multiple cloud resource administrators
separation of duties and split keys

4 Hybrid Backend

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 application is hosted in a static data center.

4.1 Problem

How can Processing Components that experience varying workload and need access to large amounts of data be hosted in an elastic environment while the remainder of the application is hosted in a static environment?

4.2 Context

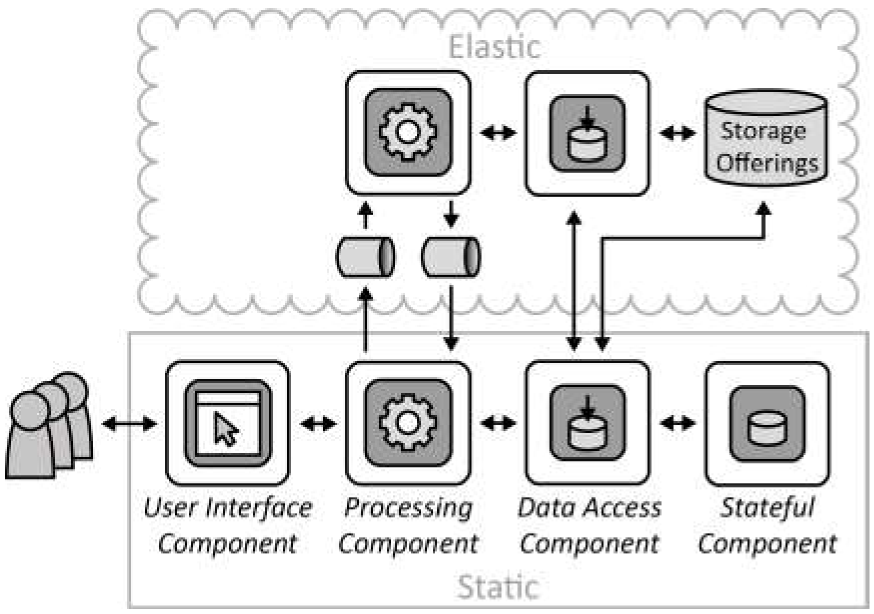
A Distributed Application provides processing functionality that experiences varying workload behavior. Mainly, Static Workload has to be handled, but some Processing Components experience Periodic Workload, Unpredictable Workload, or Continuously Changing Workload. Application components providing the respective processing functionality experiencing varying workload should, therefore, be hosted in an elastic environment. However, these components have to access large amounts of data during their execution making them highly dependent on the availability and the timely access to such data.

4.3 Solution

The Processing Components experiencing varying workloads are hosted in an elastic cloud together with the data accessed during their operation. Processing Components in the elastic cloud are triggered from the static environment through asynchronous messages exchanged via message queues provided by a Message-oriented Middleware. A Data Access Component in the static environment ensures that data required by elastic Processing Components is stored in Storage Offerings The location where this data is stored may then be passed to the elastic Processing Components during their enactment via messages. Data that is not required by the backend functionality may still be stored in Stateful Components hosted in the static data center.

Source: Hybrid Backend

4.4 MODEL



5 MULTIPATH RESOURCE ACCESS

When the path to an IT resource is lost or becomes unavailable, the IT resource becomes inaccessible. This can jeopardize the stability of an entire cloud-based solution until the cloud provider is able to supply the cloud consumer with the lost or updated path.

5.1 PROBLEM

How can an IT resource be accessed when its pre-defined path is lost or becomes unavailable?

5.2 CONTEXT

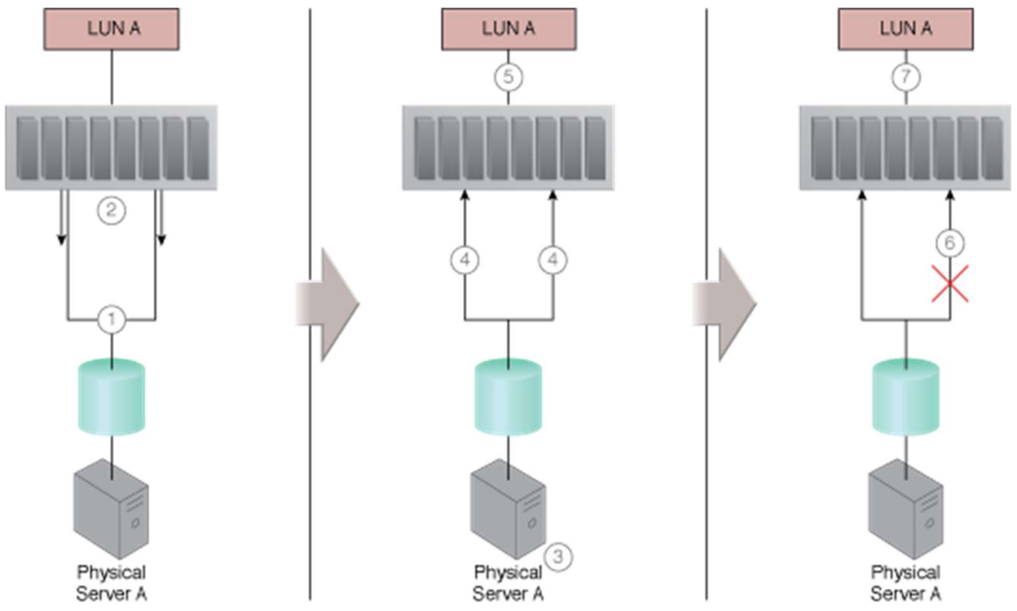
Alternative paths to IT resources are provided to give cloud consumers a means of programmatically or manually overcoming path failures.

5.3 SOLUTION

A multipathing system that resides on the server or hypervisor is established to provide multiple alternative paths to the same, unique IT resource, while ensuring that the IT resource is viewed identically via each alternative path.

Source: Multipath Resource Access

5.4 Model



6 Resource Pooling

When sharing identical IT resources for scalability purposes, it can be error-prone and burdensome to keep them fully synchronized on an on-going basis.

6.1 Problem

How can IT resources be organized to support dynamic sharing?

6.2 Context

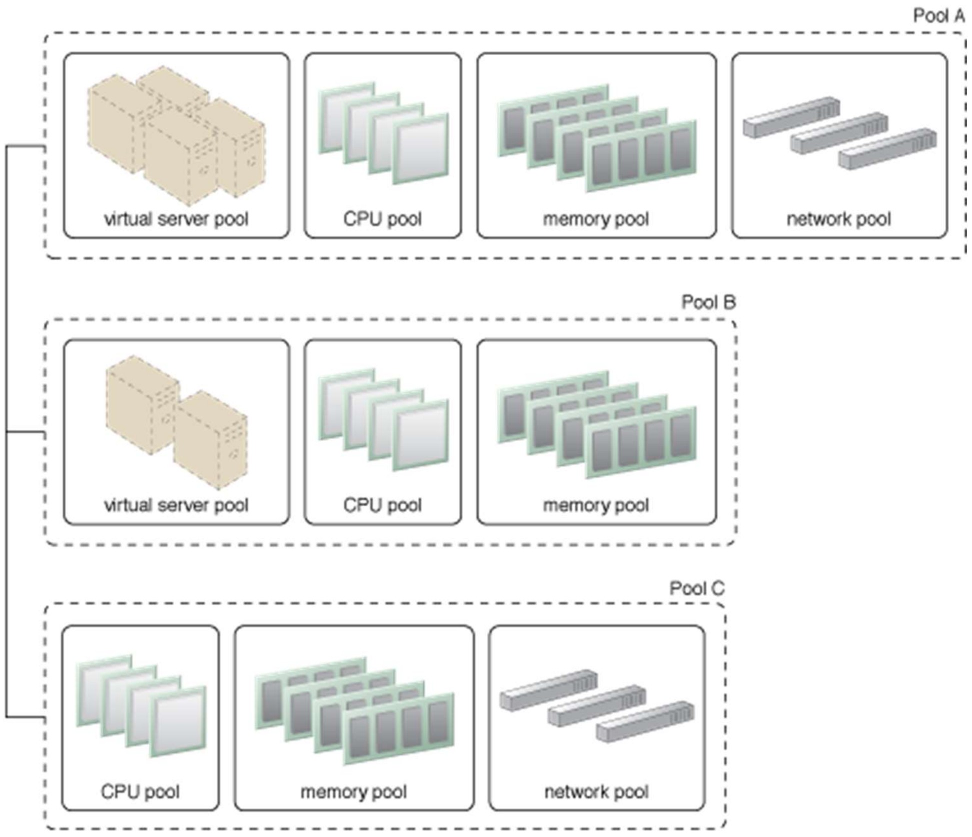
An automated synchronization system is provided to group identical IT resources into pools and to maintain their synchronicity.

6.3 Solution

Resource pools can be created at different sizes and further organized into hierarchies to provide parent and child pools.

Source: Resource Pooling

6.4 MODEL



7 COMPLIANT DATA REPLICATION

Data is replicated among multiple environments that may handle different data subsets. During replication data is obfuscated and deleted depending on laws and security regulations. Data updates are adjusted automatically to reflect the different data structures handled by environments.

7.1 PROBLEM

How can data be replicated between environments if some environments may only handle subsets of the data due to laws and corporate regulations?

7.2 CONTEXT

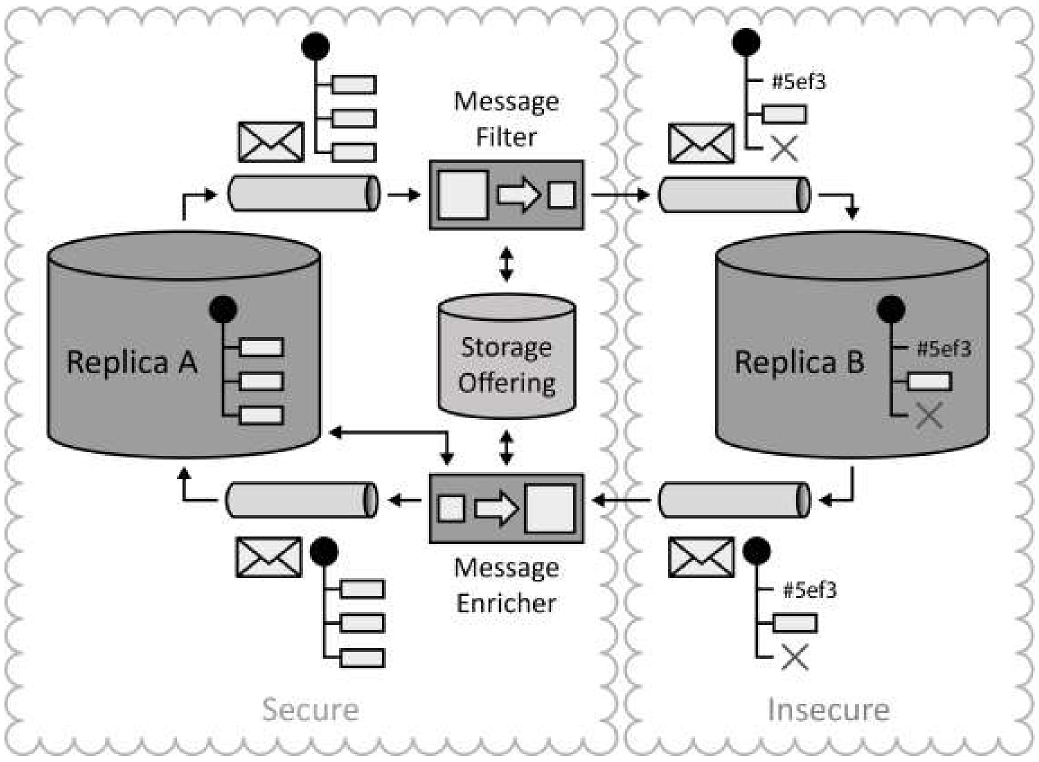
Distributed Applications that are hosted in a Hybrid Cloud often require access to the same data from different application components. If application components accessing the data are globally distributed, data access performance may be reduced drastically if data is only stored in one geographic location. Therefore, data may have to be replicated. Due to laws and corporate regulations, some of these locations may only handle a subset of the available data or data has to be obfuscated.

7.3 SOLUTION

Data replicas in different environments are updated asynchronously using messaging. Message filters are used to delete and obfuscate certain data elements in these messages as they leave the trusted environment. Information about the data manipulations stored in a storage offering. If data is then altered in the less secure environment, the corresponding update message is enriched by a message enricher as it enters the secure environment.

Source: Compliant Data Replication

7.4 MODEL



8 DYNAMIC DATA NORMALIZATION

Cloud consumers may store large volumes of redundant data within cloud storage devices, thereby bloating the storage architecture and compromising data access performance.

8.1 PROBLEM

How can redundant data within cloud storage devices be automatically avoided?

8.2 CONTEXT

Data received by cloud consumers is automatically normalized so that redundant data is avoided and cloud storage device capacity and performance is optimized.

8.3 Solution

Data de-duplication technology is used to detect and eliminate redundant data at block or file-based levels.

9 Cloud Storage Data Lifecycle Management

Datasets no longer required by an organization can bloat databases causing performance challenges, and can further incur administration and maintenance burdens.

9.1 Problem

How can data be stored and managed in a cloud environment based on a defined lifecycle?

9.2 Context

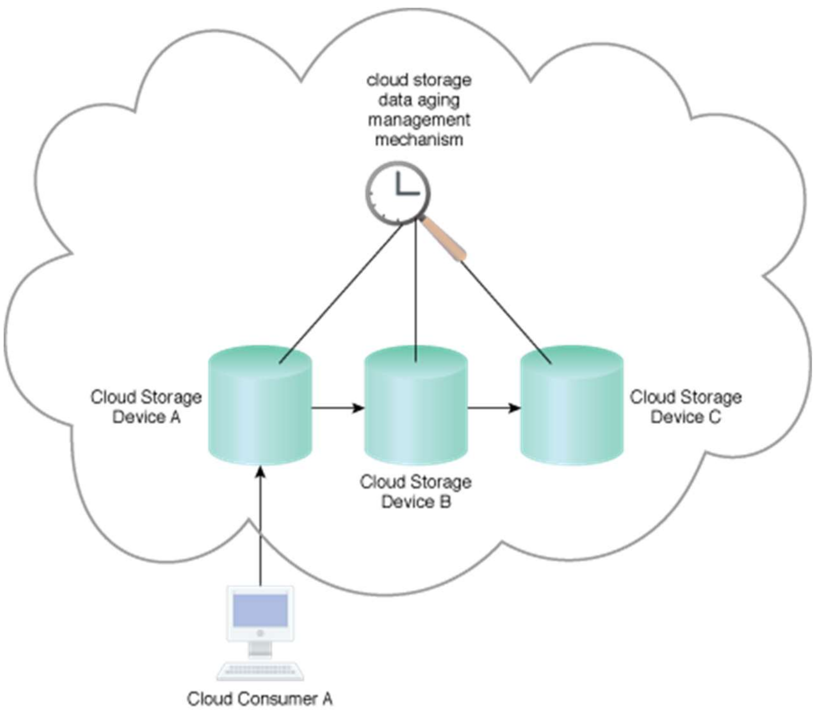
A solution can be introduced to automatically manage and migrate the data into a different type of cloud storage device, or delete the data based on its state in a defined lifecycle.

9.3 Solution

A cloud storage data aging management mechanism monitors the state of data against a provided lifecycle in order to move data to a different cloud storage device or delete data after a defined lifecycle.

Source: Cloud Storage Data Lifecycle Management

9.4 Model



10 Elasticity Manager

The utilization of IT resources on which an elastically scaled-out application is hosted, for example, virtual servers is used to determine the number of required application component instances.

10.1 Problem

How can the number of required application component instances be determined based on the utilization of hosting IT resources?

10.2 Context

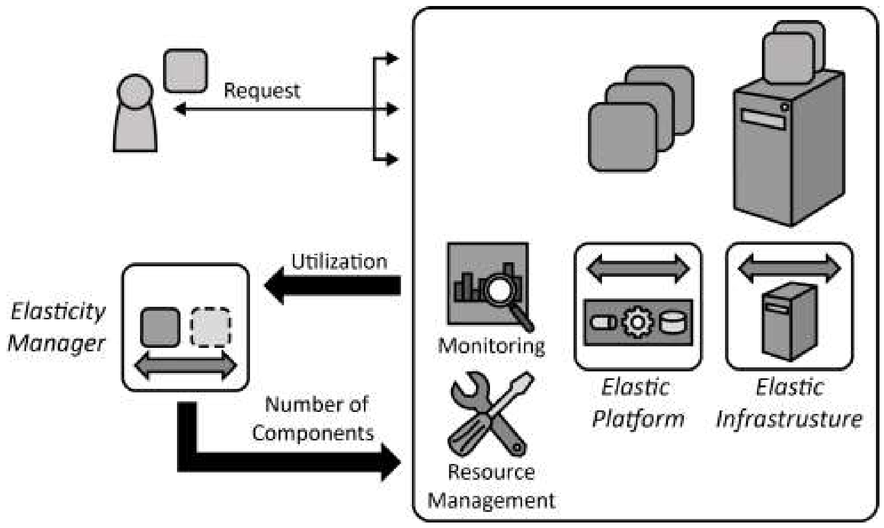
Application components of a Distributed Application hosted on an Elastic Infrastructure or Elastic Platform shall be scaled-out. The instances of applications components, thus, shall be provisioned and decommissioned automatically based on the current workload experienced by the application.

10.3 Solution

The utilization of cloud resources on which application component instances are deployed is monitored. This could be, for example, the CPU load of a virtual server. This information is used to determine the number of required instances.

Source: Elasticity Manager

10.4 Model



11 Two-Tier Cloud Application

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

11.1 Problem

Why separate a cloud application into stateful (data handling) and stateless components?

11.2 Context

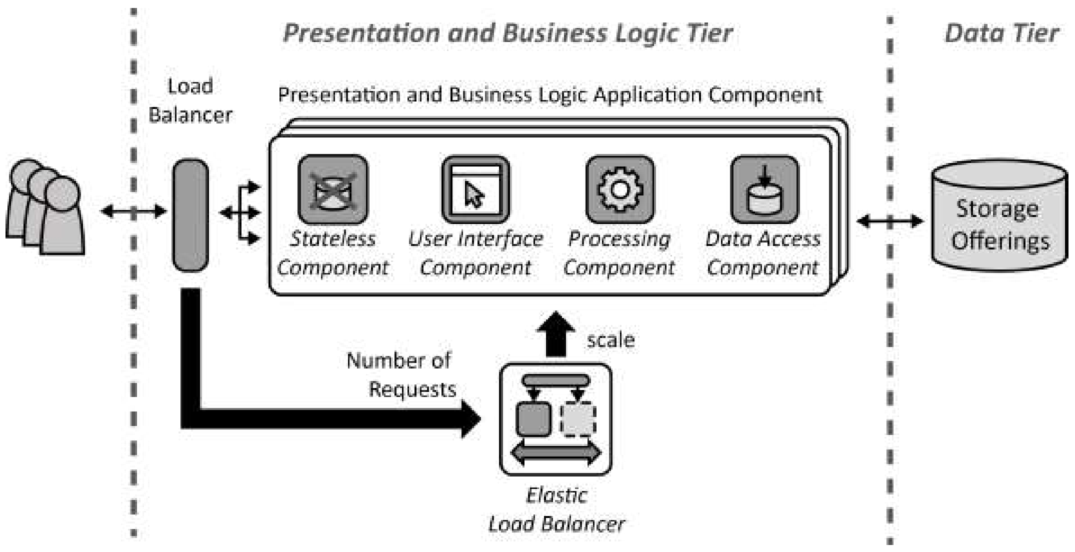
A Distributed Application is composed of multiple application components independently scalable. Data handling or stateful functionality is significantly harder to scale than stateless components, since stateful components have to provide state information between instances. Therefore, a cloud application should be composed of easy-to-scale, stateless and hard-to-scale, stateful components.

11.3 Solution

The cloud application's functionality is decomposed into data handling, stateful components, accompanied by one or several storage offerings, and stateless components handling presentation and business logic. This separation enables the two tiers to elastically and independently adapt to their workload.

Source: Two-Tier Cloud Application

11.4 Model



12 Micro Scatter-Gather

A high-performance task needs to be completed that would need to involve the composition of multiple cloud services, as well as the composition of specific cloud service instances. The systems programming required to build such composition logic is complex.

12.1 Problem

How can a cloud service carry out high-performance composition logic that can include composing specific cloud service instances?

12.2 Context

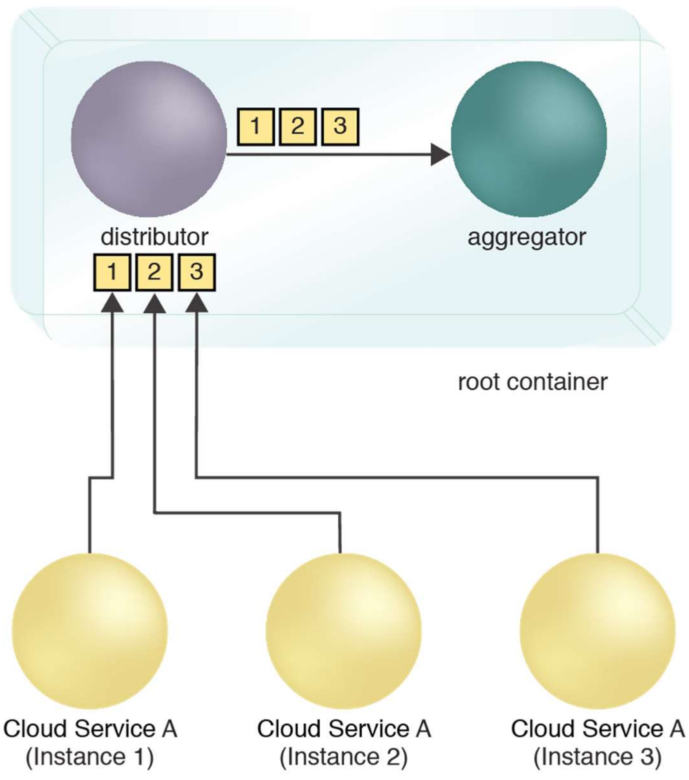
A root container is utilized with special distributor and aggregator cloud services designed to compose and interact with multiple cloud services and cloud service instances, thereby carrying out the necessary high-performance composition logic.

12.3 Solution

The distributor cloud service accepts the task from the service consumer and segregates it into micro tasks that are assigned to the appropriate cloud services or cloud service instances. A separate aggregator service collects the results and aggregates them into one single response back to the service consumer. Distributor and aggregator services are deployed in isolation in a special root container that exposes APIs to the service consumer and the composed cloud services.

Source: Micro Scatter-Gather

12.4 Model


1 2 3
distributor
1 2 3
aggregator
root container
Cloud Service A (Instance 1)
Cloud Service A (Instance 2)
Cloud Service A (Instance 3)

13 Geotagging

Control of data, workload locality and data sovereignty are requirements for federal and industrial compliance and regulatory issues. Without knowledge of where a cloud consumer's workload is executed, it may not be possible to meet industrial compliance and regulatory requirements for data sovereignty.

13.1 Problem

How can the geographic location of cloud-based data and workloads be automatically communicated?

13.2 Context

When trusted resource pools are generated, the geolocation is supplied as part of the compliance and regulatory assurance attributes.

13.3 Solution

Assets are geotagged using TPMs and are added to trusted resource pools that contain the same security assurance levels and geolocation regions.

Source: Geotagging

14 Loose Coupling

A communication intermediary separates application functionality from concerns of communication partners regarding their location, implementation platform, the time of communication, and the used data format.

14.1 Problem

How can dependencies between Distributed Applications and between individual components of these applications be reduced?

14.2 Context

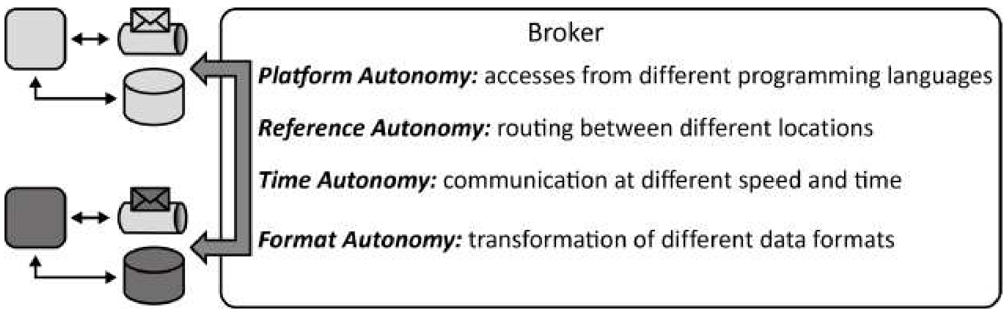
Information exchange between applications and their individual components as well as associated management tasks, such as scaling, failure handling, or update management can be simplified significantly if application components can be treated individually and the dependencies among them are kept to a minimum.

14.3 Solution

Communicating components and multiple integrated applications are decoupled from each other by interacting through a broker. This broker encapsulates the assumptions that communication partners would otherwise have to make about one other and, thus, ensures separation of concerns.

Source: Loose Coupling Pattern

14.4 Model


Broker
Platform Autonomy: accesses from different programming languages
Reference Autonomy: routing between different locations
Time Autonomy: communication at different speed and time
Format Autonomy: transformation of different data formats

15 Provider Adapter

Provider interfaces are encapsulated and mapped to unified interfaces used in applications to separate concerns of interactions with the provider from application functionality.

15.1 Problem

How can the dependencies of an application component on a provider-specific interface be managed?

15.2 Context

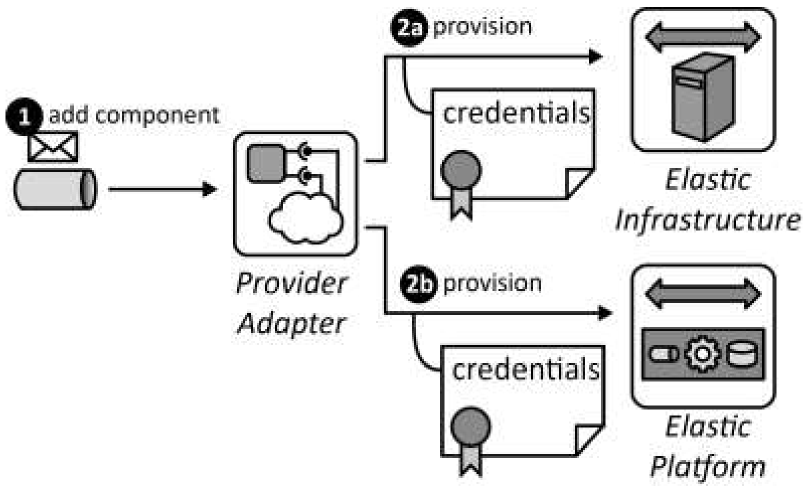
Cloud providers offer many interfaces that can be used in application components of a Distributed Application. If a component directly interacts with these interfaces, its implementation becomes strongly interleaved with the specific functions offered and the protocols used.

15.3 Solution

The Provider Adapter encapsulates all provider-specific implementations required for authentication, data formatting etc. in an abstract interface. The Provider Adapter , thus, ensures separation of concerns between application components accessing provider functionality and application components providing application functionality. It may also offer synchronous provider-interfaces to be accessed asynchronously via messages and vice versa.

Source: Provider Adapter

15.4 Model



16 Elastic Load Balancer

The number of synchronous accesses to an elastically scaled-out application is used to determine the number of required application component instances.

16.1 Problem

How can the number of required application component instances be determined based on monitored synchronous accesses?

16.2 Context

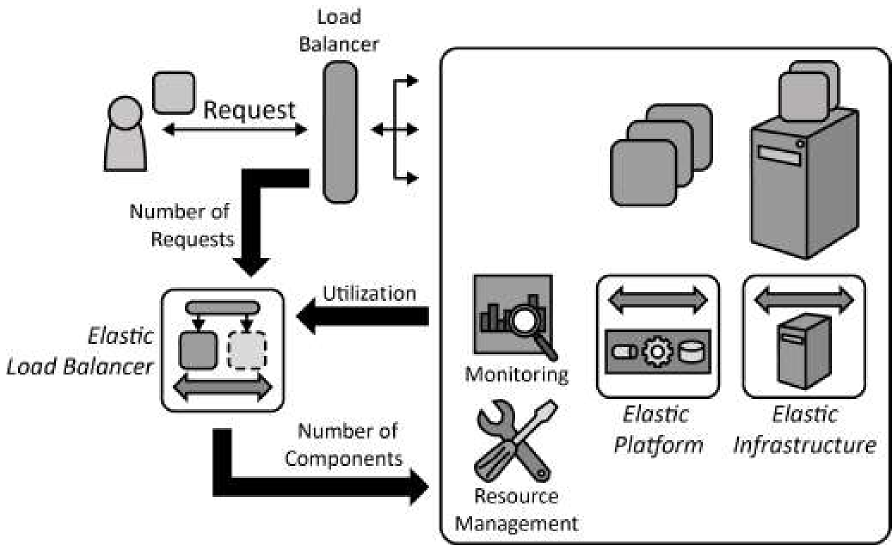
Application components of a Distributed Application shall be scaled out automatically. Requests sent to an application shall be used as an indicator for the currently experienced workload from which the required number of components instances shall be deducted.

16.3 Solution

Based on the number of synchronous requests handled by a load balancer and possibly other utilization information, the required number of required component instances is determined.

Source: Elastic Load Balancer

16.4 Model



17 Hybrid User Interface

Varying workload from a user group interacting asynchronously with an application is handled in an elastic environment while the remainder of an application resides in a static environment.

17.1 Problem

How can a user interface for asynchronous interaction be hosted in a cloud while being integrated with an application otherwise hosted in a static data center?

17.2 Context

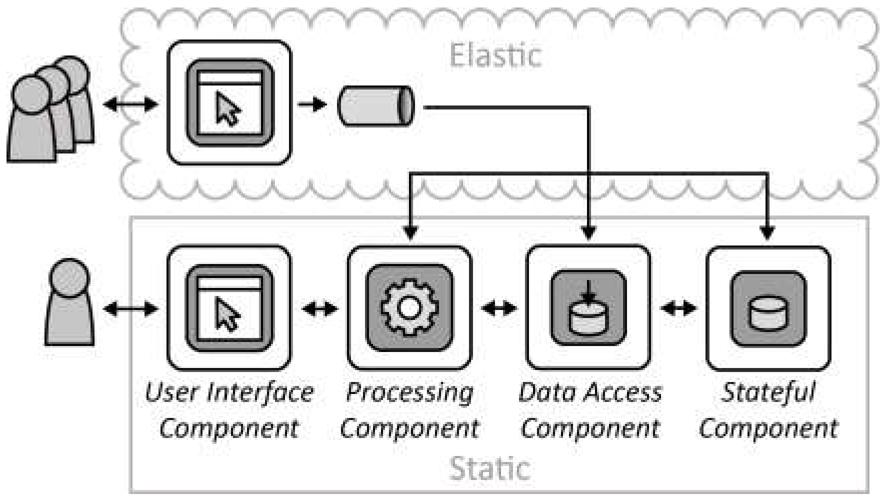
An application serves user groups with different workload behavior. One user group generates Static Workload, while another user group generates Periodic Workload, Once-in-a-lifetime Workload, Unpredictable Workload, or Continuously Changing Workload. Since the predictability of the user group size and workload behavior differs, it shall be ensured that unexpected peak workloads do not affect the performance of the application while each user group is handled by the most suitable cloud environment.

17.3 Solution

The User Interface Component serving users generating varying workload is hosted in an elastic cloud environment. Other application components that are hosted in a static environment. The user interface deployed in the elastic cloud is integrated with the remainder of the application in a decoupled fashion using messaging to ensure Loose Coupling.

Source: Hybrid User Interface

17.4 Model



18 In-Transit Cloud Data Encryption

Data copied to and from a cloud environment transits networks and servers beyond the control of the organization and can be intercepted by malicious intermediaries.

18.1 Problem

How can data be securely transmitted to, from, and within a cloud environment?

18.2 Context

A solution is implemented with capabilities that secure and protect data while it transfers between sender and receiver and also ensure that data will not be accepted by the receiver if the original data sent is modified.

18.3 Solution

An encryption mechanism is implemented to encrypt data between sender and receiver for confidentiality, and a digital signature mechanism is implemented to provide integrity for the data.

Source: In-Transit Cloud Data Encryption

19 Health Endpoint Monitoring

It's a good practice, and often a business requirement, to monitor web applications and back-end services, to ensure they're available and performing correctly. However, it's more difficult to monitor services running in the cloud than it is to monitor on-premises services. For example, you don't have full control of the hosting environment, and the services typically depend on other services provided by platform vendors and others. There are many factors that affect cloud-hosted applications such as network latency, the performance and availability of the underlying compute and storage systems, and the network bandwidth between them. The service can fail entirely or partially due to any of these factors. Therefore, you must verify at regular intervals that the service is performing correctly to ensure the required level of availability, which might be part of your service level agreement (SLA).

19.1 Problem

Implement functional checks in an application that external tools can access through exposed endpoints at regular intervals

19.2 Context

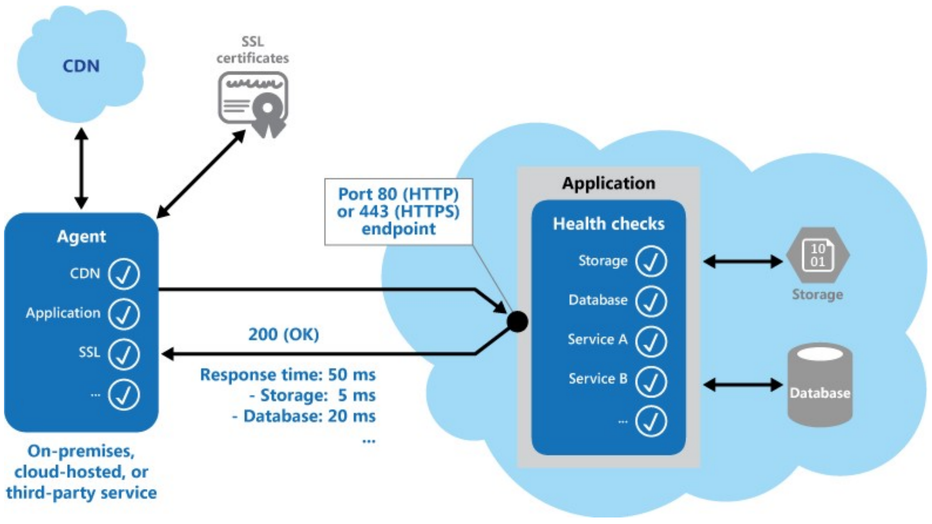
This pattern is useful for: Monitoring websites and web applications to verify availability. Monitoring websites and web applications to check for correct operation. Monitoring middle-tier or shared services to detect and isolate a failure that could disrupt other applications. Complementing existing instrumentation in the application, such as performance counters and error handlers. Health verification checking doesn't replace the requirement for logging and auditing in the application. Instrumentation can provide valuable information for an existing framework that monitors counters and error logs to detect failures or other issues. However, it can't provide information if the application is unavailable.

19.3 Solution

Implement health monitoring by sending requests to an endpoint on the application. The application should perform the necessary checks, and return an indication of its status. A health monitoring check typically combines two factors: The checks (if any) performed by the application or service in response to the request to the health verification endpoint. Analysis of the results by the tool or framework that performs the health verification check. The response code indicates the status of the application and, optionally, any components or services it uses. The latency or response time check is performed by the monitoring tool or framework. The figure provides an overview of the pattern.

Source: [Health Endpoint Monitoring pattern]()

19.4 Model



20 Hybrid Data

Data of varying size is hosted in an elastic cloud while the remainder of an application resides in a static environment.

20.1 Problem

How can a data handling functionality that experiences varying workload be hosted in an elastic cloud while the rest of the application is located in a static data center?

20.2 Context

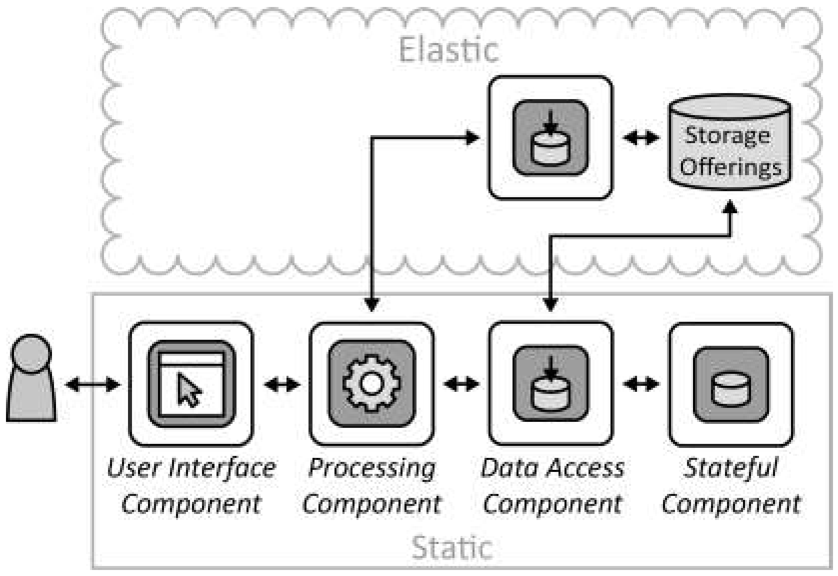
A Distributed Application handles data whose size varies drastically over time. Large amounts of data may, thus, be generated periodically and are then deleted again, may increase and decrease randomly, or may display a general increase or decrease over time. Especially, during these changes, the user number and their accesses to the application can be static resulting in Static Workload on the remainder of the application components.

20.3 Solution

Data whose varying size makes it unsuitable for hosting in a static environment is handled by Storage Offerings in an elastic cloud. At this location data is either accessed by Data Access Components that are hosted in the static environment or by Data Access Components hosted in the elastic environment.

Source: Hybrid Data

20.4 Model



21 Direct I/O Access

Virtualized networks and associated virtualized IT resources have capacity limitations that can unreasonably inhibit virtual server communication and data transfer performance.

21.1 Problem

How can a virtual server overcome data transfer capacity thresholds imposed by its surrounding virtualization environment?

21.2 Context

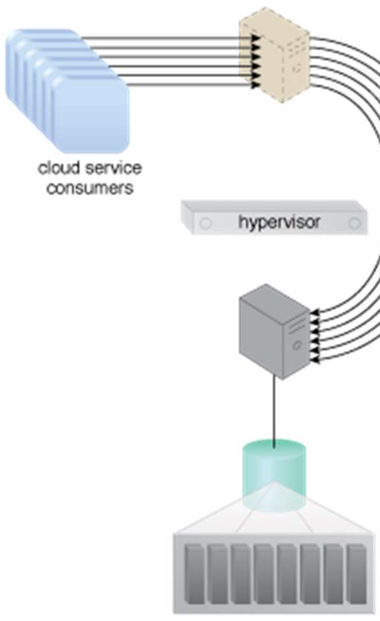
The virtual server is allowed to circumvent the hypervisor and directly access the physical server's I/O card.

21.3 Solution

The hypervisor transfers complete control of the physical server's I/O card directly to the virtual server, which is then able to recognize the I/O card as a hardware device.

Source: Direct I/O Access

21.4 Model



22 Service Registry

Implement a service registry, which is a database of services, their instances and their locations. Service instances are registered with the service registry on start-up and deregistered on shutdown. Client of the service and/or routers query the service registry to find the available instances of a service. A service registry might invoke a service instance's health check API to verify that it is able to handle requests.

22.1 Problem

How are services packaged and deployed?

22.2 Context

You have applied the Microservice architecture pattern and architected your system as a set of services. Each service is deployed as a set of service instances for throughput and availability.

22.3 Solution

Package the service as a (Docker) container image and deploy each service instance as a container

Source: Service Registry

23 SYNCHRONIZED OPERATING STATE

A cloud consumer may be prevented from utilizing high availability and clustering technology for its virtual servers or operating systems, thereby making them more vulnerable to failure.

23.1 PROBLEM

How can the availability and reliability of virtual servers be ensured when high availability and clustering technology is unavailable?

23.2 CONTEXT

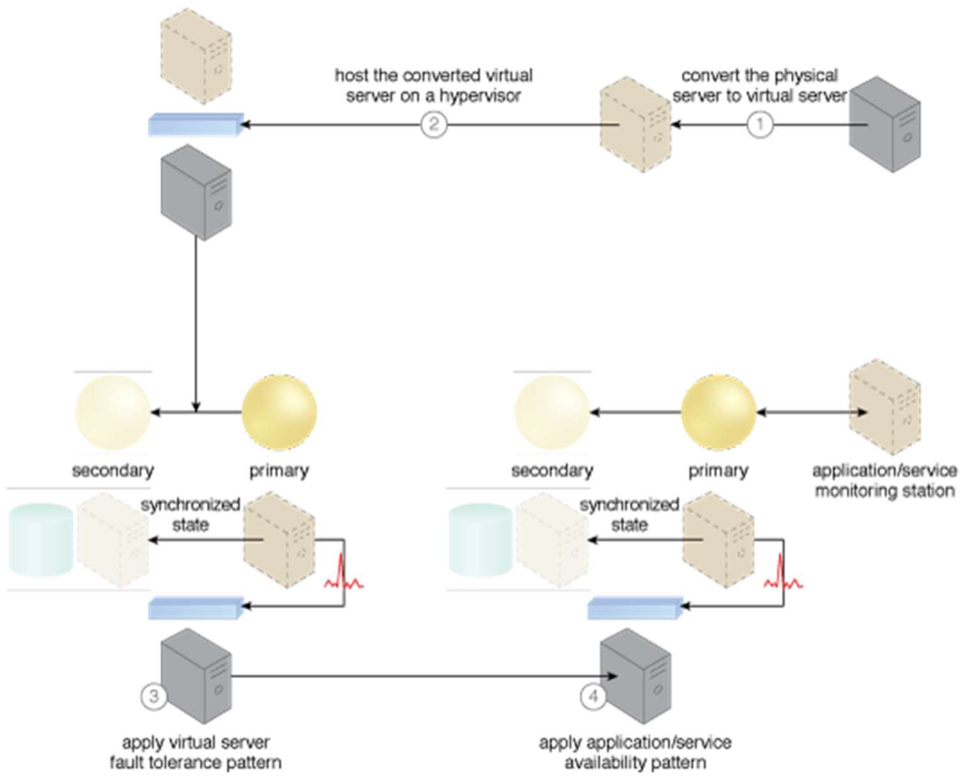
A composite failover system is created to not rely on clustering or high availability features but instead use heartbeat messages to synchronize virtual servers.

23.3 SOLUTION

The heartbeat messages are processed by a specialized service agent and are exchanged between hypervisors, the hypervisor and virtual server, and the hypervisor and VIM.

Source: Synchronized Operating State

23.4 MODEL


host the converted virtual server on a hypervisor
2
convert the physical server to virtual server
1
secondary
primary
synchronized state
secondary
primary
application/service monitoring station
synchronized state
3
apply virtual server fault tolerance pattern
4
apply application/service availability pattern

24 Cloud Storage Device Masking

Data stored in a shared cloud environment can be vulnerable to unauthorized access by other cloud consumers.

24.1 Problem

How can data stored on a cloud storage device be isolated to specific consumers?

24.2 Context

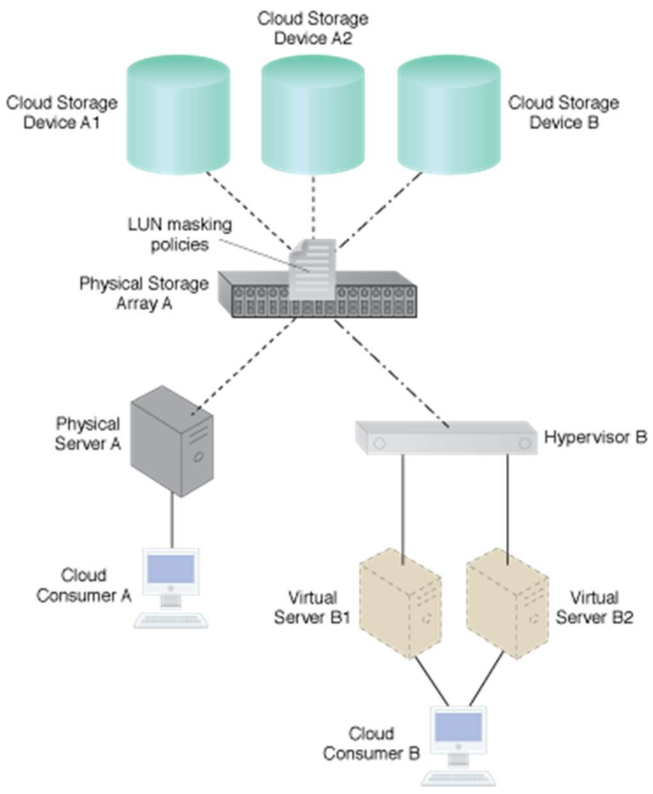
A solution is implemented to isolate each cloud storage device from being presented to or accessed by unauthorized cloud consumers.

24.3 Solution

A LUN masking mechanism can enforce defined policies at the physical storage array in order to prevent unauthorized cloud consumers from accessing a specific cloud storage device in a shared cloud environment.

Source: Cloud Storage Device Masking

24.4 Model



25 PROCESSING COMPONENT

Possibly long running processing functionality is handled by separate components to enable elastic scaling. Processing functionality is further made configurable to support different customer requirements.

25.1 PROBLEM

How can processing be scaled out elastically among distributed resources while being configurable regarding the supported functions to meet different customers requirements?

25.2 CONTEXT

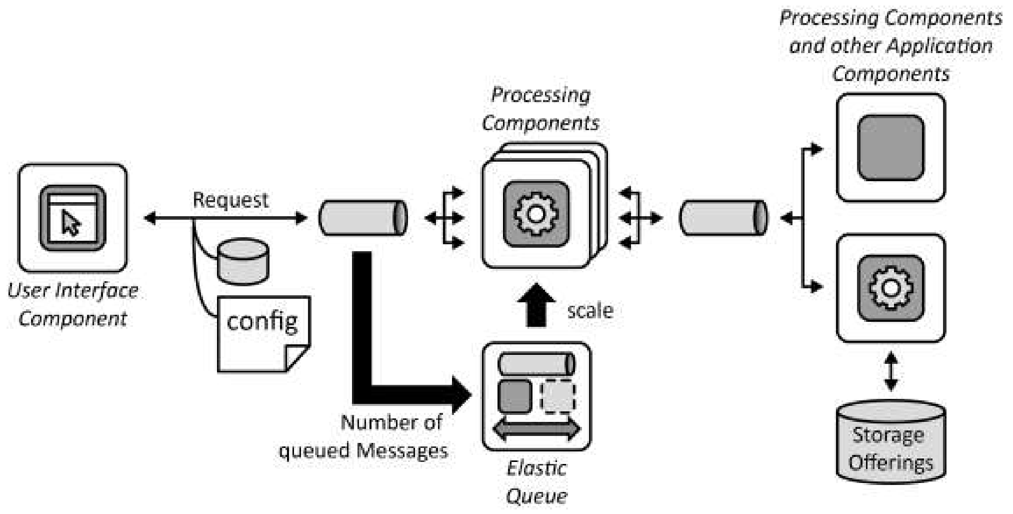
The processing functionality offered by an application shall be handled by different application component instances that operate independently. Instances of these components have to be added and removed easily to the application as part of scaling operations.

25.3 SOLUTION

Processing functionality is split into separate function blocks and assigned to independent Processing Components. Each processing component is scaled out independently and is implemented in a stateless fashion as described in the Stateless Component pattern. Scaling is handled by an Elastic Queue. Data required for processing is provided with requests or by Storage Offerings.

Source: Processing Component

25.4 MODEL



26 Content Distribution Network

Applications component instances and data handled by them are globally distributed to meet the access performance required by a global user group.

26.1 Problem

26.2 Context

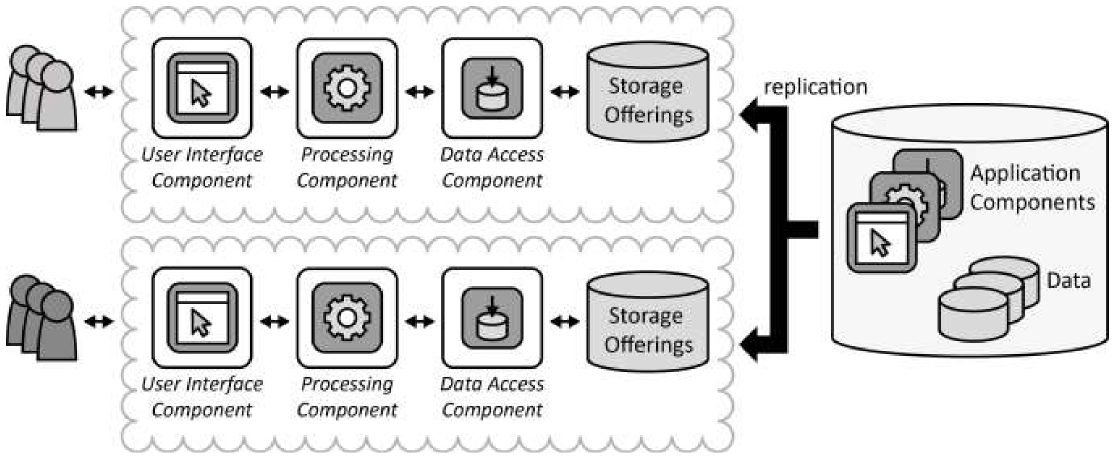
If the application provides multimedia content to users, for example, streamed videos and music the amount of data to be served increases drastically. If such multimedia content is located too far from the user accessing it, the communication delay of the distribution network may hinder the timely access to data. Therefore, storing content in only one centralized location, i.e., one cloud or data center is unfeasible.

26.3 Solution

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

Source: Content Distribution Network

26.4 Model



27 Throttling

The load on a cloud application typically varies over time based on the number of active users or the types of activities they are performing. For example, more users are likely to be active during business hours, or the system might be required to perform computationally expensive analytics at the end of each month. There might also be sudden and unanticipated bursts in activity. If the processing requirements of the system exceed the capacity of the resources that are available, it'll suffer from poor performance and can even fail. If the system has to meet an agreed level of service, such failure could be unacceptable. There're many strategies available for handling varying load in the cloud, depending on the business goals for the application. One strategy is to use autoscaling to match the provisioned resources to the user needs at any given time. This has the potential to consistently meet user demand, while optimizing running costs. However, while autoscaling can trigger the provisioning of additional resources, this provisioning isn't immediate. If demand grows quickly, there can be a window of time where there's a resource deficit.

27.1 Problem

Control the consumption of resources used by an instance of an application, an individual tenant, or an entire service. This can allow the system to continue to function and meet service level agreements, even when an increase in demand places an extreme load on resources.

27.2 Context

Use this pattern: To ensure that a system continues to meet service level agreements. To prevent a single tenant from monopolizing the resources provided by an application. To handle bursts in activity. To help cost-optimize a system by limiting the maximum resource levels needed to keep it functioning.

27.3 Solution

An alternative strategy to autoscaling is to allow applications to use resources only up to a limit, and then throttle them when this limit is reached. The system should monitor how it's using resources so that, when usage exceeds the threshold, it can throttle requests from one or more users. This will enable the system to continue functioning and meet any service level agreements (SLAs) that are in place. For more information on monitoring resource usage, see the Instrumentation and Telemetry Guidance. The system could implement several throttling strategies, including: Rejecting requests from an individual user who's already accessed system APIs more than n times per second over a given period of time. This requires the system to meter the use of resources for each tenant or user running an application. For more information, see the Service Metering Guidance. Disabling or degrading the functionality of selected nonessential services so that essential services can run unimpeded with sufficient resources. For example, if the application is streaming video output, it could switch to a lower resolution. Using load leveling to smooth the volume of activity (this approach is covered in more detail by the Queue-based Load Leveling pattern). In a multi-tenant environment, this approach will reduce the performance for every tenant. If the system must support a mix of tenants with different SLAs, the work for high-value tenants might be performed immediately. Requests for other tenants can be held back, and handled when the backlog has eased. The Priority Queue pattern could be used to help implement this approach. Deferring operations being performed on behalf of lower priority applications or tenants. These operations can be suspended or limited, with an exception generated to inform the tenant that the system is busy and that the operation should be retried later.

Source: Throttling pattern

28 Service State Management

A cloud service designed to place significant data into memory for prolonged periods can consume excessive amounts of runtime processing, thereby unreasonably taxing the overall cloud infrastructure and imposing additional usage costs on cloud consumers.

28.1 Problem

How can stateful cloud services be optimized to minimize runtime IT resource consumption?

28.2 Context

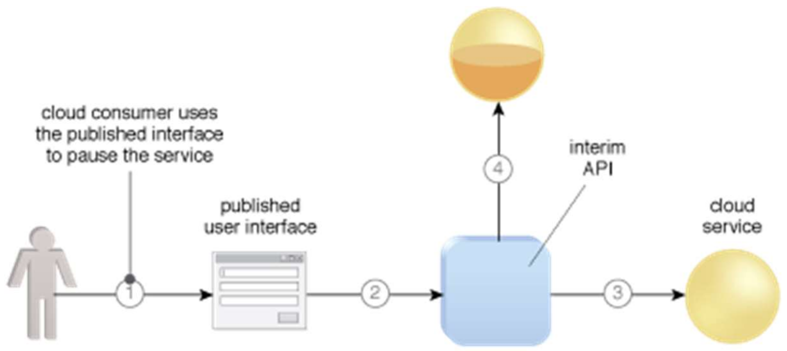
The cloud service is designed to integrate with a state management system allowing it to defer state data at runtime when necessary so as to minimize its IT resource consumption.

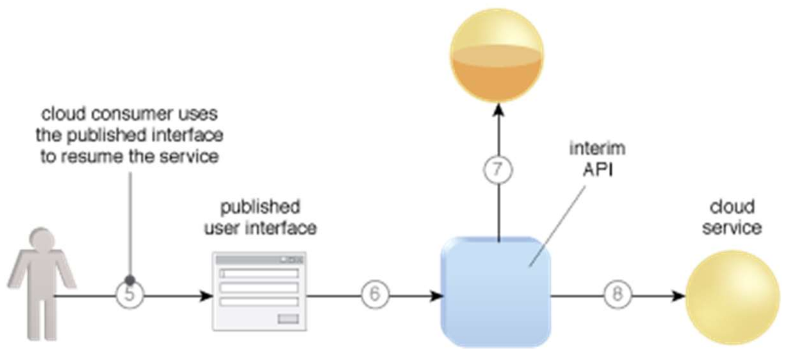
28.3 Solution

A state management system requires a cloud storage device capable of temporarily holding and releasing state data exchanged by the cloud service. The cloud service itself needs to be equipped with logic to determine when and how to release and retrieve state data.

Source: Service State Management

28.4 Model





29 Cloud Storage Device Performance Enforcement

A cloud consumer can have different performance requirements for different datasets. If datasets with varying performance requirements reside on a cloud storage device with fixed performance capabilities, the performance requirements will not be met.

29.1 Problem

How can data with different performance characteristics be stored on a cloud storage device compliant with the performance requirements of each dataset?

29.2 Context

A solution is implemented with the ability to match and compare the performance characteristics of datasets against performance capabilities of a destination cloud storage device.

29.3 Solution

A cloud storage device performance monitor mechanism manages data stored on a cloud storage device based on performance characteristics. This solution can also enforce policies that prevent data from being copied or moved elsewhere while sending an alert response.

Source: Cloud Storage Device Performance Enforcement

30 Hybrid Backup

Data is periodically extracted from an application to be archived in an elastic cloud for disaster recovery purposes.

30.1 Problem

How can data be archived in a remote environment while the remainder of the application is hosted in a static environment?

30.2 Context

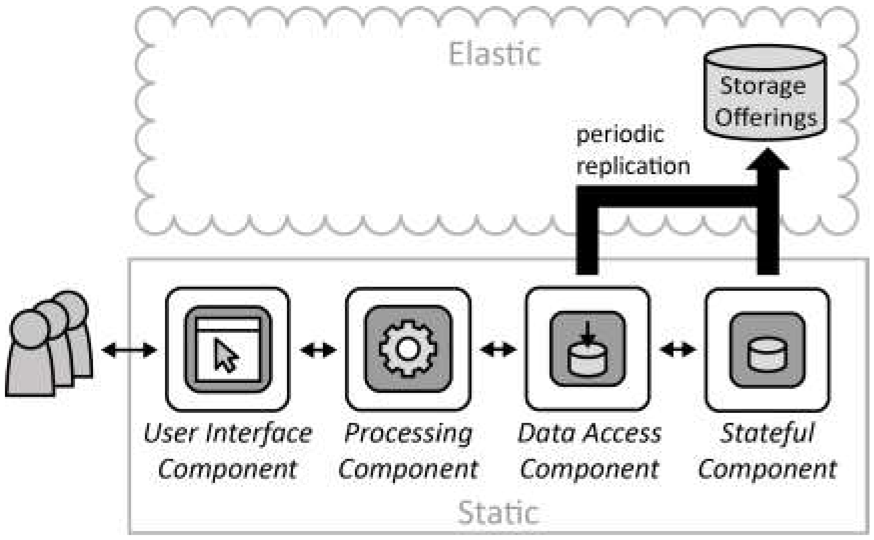
Many applications are used by small and medium businesses which do not have the required IT skills to host and maintain their own highly available infrastructure. Especially, requirements regarding business resiliency * the ability to recover from an error * and business continuity * the ability to operate during an error * are challenging. Furthermore, there are laws and regulations making businesses liable to archive data and keep it accessible for audits, often over very long periods of time.

30.3 Solution

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

Source: Hybrid Backup

30.4 Model



31 Three-Tier Cloud Application

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

31.1 Problem

When to decompose presentation logic, business logic, and data handling into three separate tiers that can be scaled independently?

31.2 Context

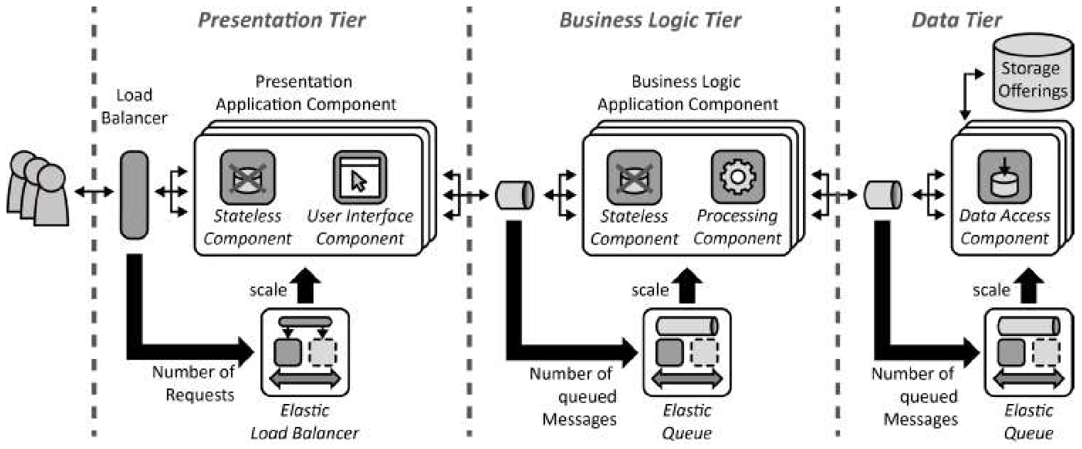
Dividing a Distributed Application into three tiers instead of two (see Two-Tier Cloud Applications) results in a more efficient application, if there are processing components (e.g. business logic) which are more computationally intensive or are used less frequently than UI components, summarizing their implementation in one tier can be inefficient. This is the case if components experience different workloads. The number of provisioned component instances cannot be aligned well to differing workloads if they are grouped into coarse grained tiers.

31.3 Solution

The application is decomposed into three tiers, where each tier can be scaled independently and elastically. The presentation tier is comprised of a load balancer and an application component that implements the Stateless Component pattern and User Interface Component pattern. The business logic tier is comprised of an application component implementing the Stateless Component pattern in addition to the Processing Component pattern.

Source: [Three-Tier Cloud Application]

31.4 Model



32 Stateless Component

The application's state is detached from the application's functionality to ease scaling-out and make the application more tolerant towards component failures.

32.1 Problem

How to increase an application component's elasticity and robustness?

32.2 Context

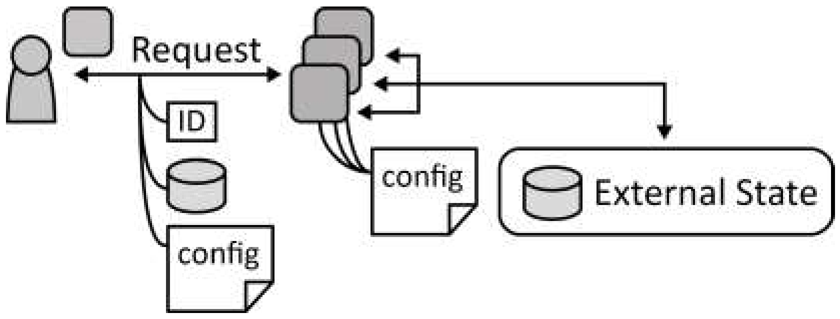
The components of a Distributed Application are distributed among multiple cloud resources or cloud providers to benefit from this distributed runtime environment. The most significant factor complicating addition and removal of such component instances is their internal state. In case of failure, this information may be lost.

32.3 Solution

Application components are implemented in a fashion that they do not have an internal state. Instead, their state and configuration is stored externally in Storage Offerings or provided to the component with each request.

Source: Stateless Component Pattern

32.4 Model



33 Cloud Resource Access Control

Proprietary cloud providers each have their own methods and protocols for authentication and access control, which contributes to vendor lock-in.

33.1 Problem

How can cloud consumer attributes be made available to determine cloud resource access control in multiple proprietary clouds?

33.2 Context

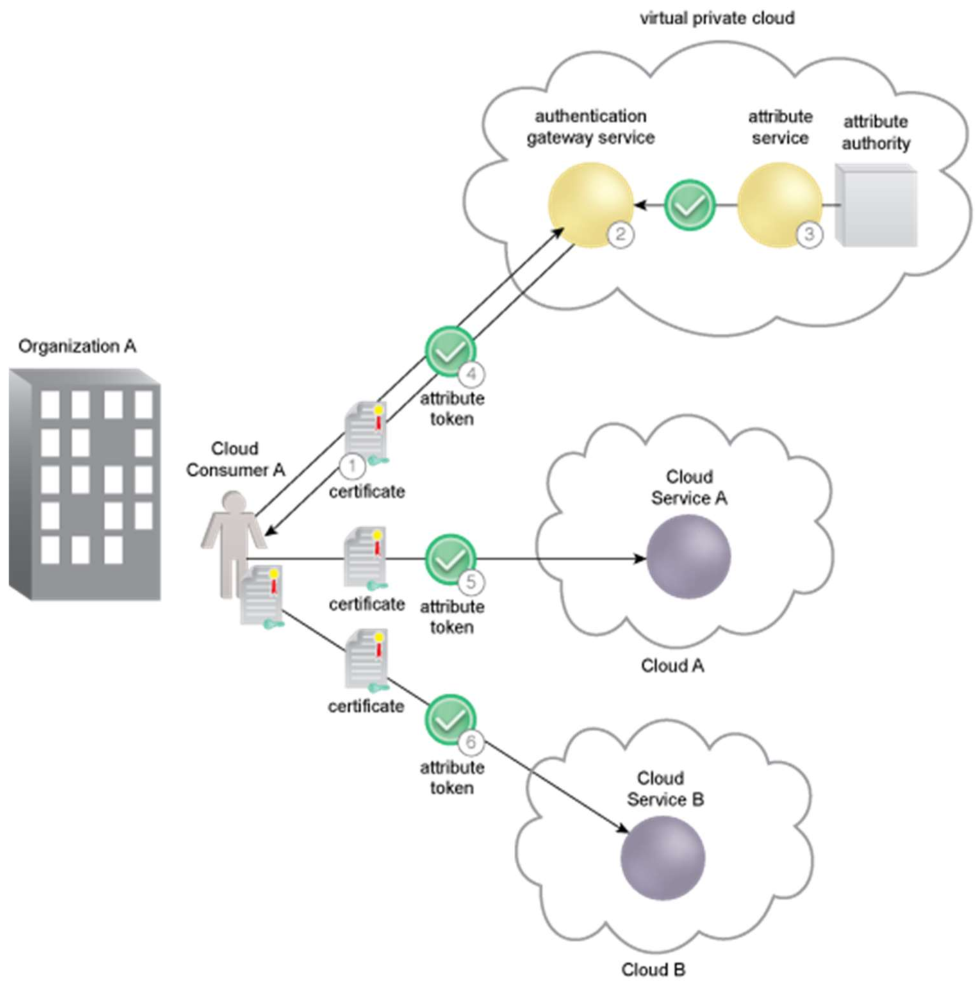
A cloud single sign-on (SSO) architecture is established, incorporating an authentication gateway service (AGS) and attribute authority for implementation of cloud resource access control.

33.3 Solution

An AGS and attribute authority connected to a secure token service (STS) are implemented and provisioned with the organization's cloud consumers' accounts and attributes. An attributed based access control (ABAC) mode of access control is instituted as the cloud service provider using the organization's SSO infrastructure.

Source: Cloud Resource Access Control

33.4 Model



34 Managed Configuration

Scaled-out application components should use a centrally stored configuration to provide a unified behavior that can be adjusted simultaneously.

34.1 Problem

How can the configuration of scaled out application component instances be controlled in a coordinated fashion?

34.2 Context

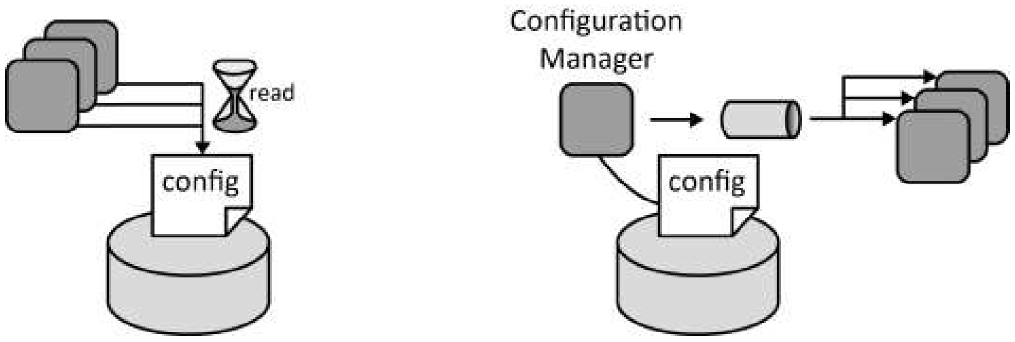
Application components of a Distributed Application often have configuration parameters. Storing configuration information together with the application component implementation can be unpractical as it results in more overhead in case of configuration changes. Each running instance of the application component must be updated separately. Component images stored in Elastic Infrastructures or Elastic Platforms also have to be updated upon configuration change.

34.3 Solution

Configurations are stored in a central Storage Offering, commonly, a Relational Database, Key-Value Storage, or Blob Storage from where it is accessed by all running component instances either by accessing the storage periodically or by sending messages to the components.

Source: Managed Configuration

34.4 Model



35 Cloud Storage Data at Rest Encryption

Data stored in a cloud environment requires security against access to the physical hard disks forming the cloud storage device.

35.1 Problem

How can cloud providers securely store cloud consumer data on cloud storage devices?

35.2 Context

Secure data on the physical hard disks in order to prevent unauthorized access.

35.3 Solution

An encryption mechanism supported by the physical storage arrays can be used to automatically encrypt data stored on the disks and decrypt data leaving the disks.

Source: Cloud Storage Data at Rest Encryption

36 Cloud Key Management

While encryption is foundational to cloud security, the management of encryption keys is one of the most difficult challenges in cloud computing. Failure to adequately manage encryption keys can lead to a range of administrative and security problems.

36.1 Problem

How can encryption keys be effectively managed for a cloud environment?

36.2 Context

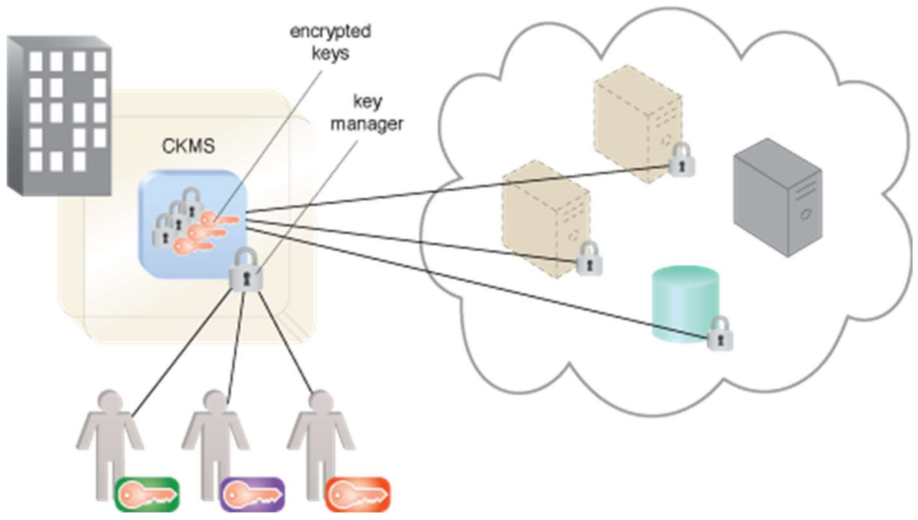
A cloud key management system is employed, available either as a physical or virtual network attached device.

36.3 Solution

A cryptographic key management system (CKMS), optionally using a hardware security module (HSM) for key protection, consisting of systems, personnel and policies is implemented to manage keys for encryption of all required data for both on-premise and cloud resources.

Source: Cloud Key Management

36.4 Model



37 Elastic Queue

The number of asynchronous accesses via messaging to an elastically scaled-out application is used to adjust the number of required application component instances.

37.1 Problem

How can the number of required application component instances be adjusted based on monitored asynchronous accesses?

37.2 Context

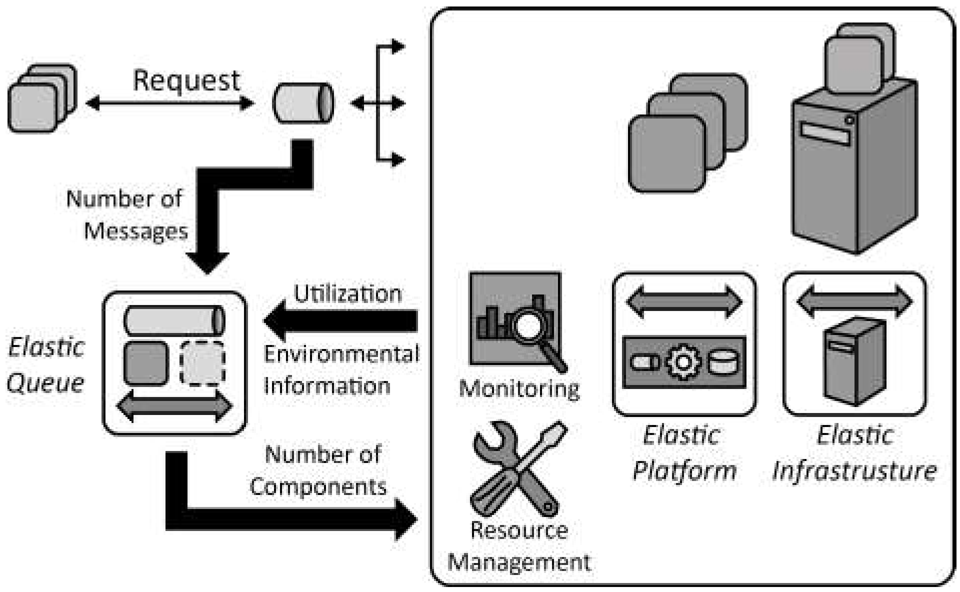
A Distributed Application is comprised of multiple application components that are accessed asynchronously and deployed to an Elastic Infrastructure or an Elastic Platform. The required provisioning and decommissioning operations to scale this application should be performed in an automated fashion.

37.3 Solution

Queues that are used to distribute asynchronous requests among multiple application components instances are monitored. Based on the number of enqueued messages the Elastic Queue adjusts the number of application component instances handling these requests.

Source: Elastic Queue

37.4 Model



38 Cloud Authentication Gateway

Cloud consumers are compelled to support multiple authentication, communication and session protocols when cloud service providers deliver components, applications, and service compositions with diverse protocol requirements.

38.1 Problem

How can cloud-based IT resources be made accessible to cloud service consumers with diverse protocol requirements?

38.2 Context

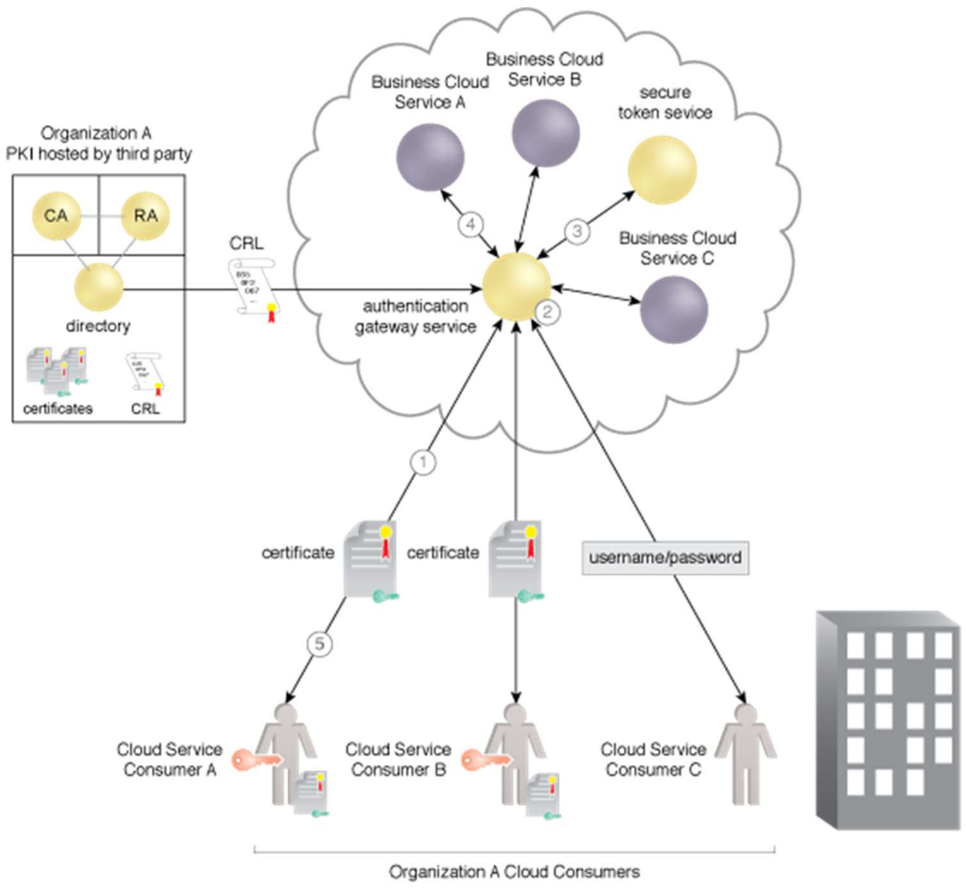
An authentication service is implemented, allowing standard authentication, communication, and session establishment from a cloud consumer to the authentication service. The authentication service then authenticates to the cloud resource on behalf of the cloud consumer using the diverse protocols required by the cloud provider.

38.3 Solution

An authentication gateway service (AGS) is established as a reverse proxy front end between the cloud consumer and the cloud resource, which intercepts and terminates the consumer's encrypted network connection, authenticates the cloud consumer, authenticates itself and the consumer to the cloud provider, and then proxies all communication between the two. All three parties are authenticated in some combination of transport level or application level communication.

Source: Cloud Authentication Gateway

38.4 Model



39 Direct LUN Access

LUNs mapped via a host bus adapter on the hypervisor can restrict data access to emulated file-based storage, which can impose performance limitations.

39.1 Problem

How can a virtual server overcome performance limitations imposed by emulated file-based storage?

39.2 Context

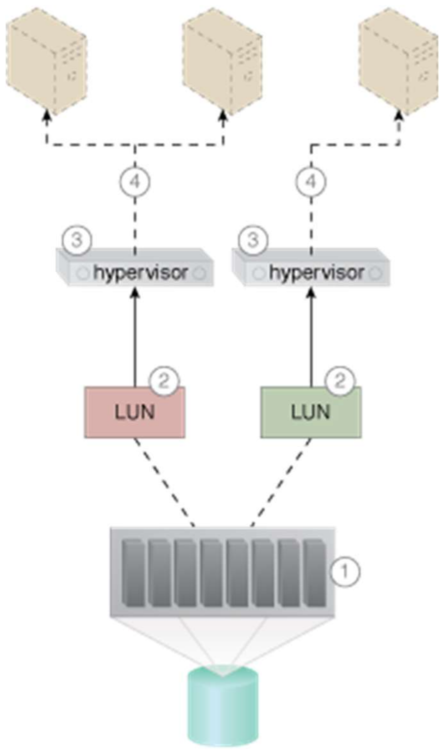
The virtual server is granted direct access to block-based storage LUNs via the physical host bust adapter card.

39.3 Solution

Raw device mapping technology is used to configure the hypervisor to enable access to raw, block-based LUNs to virtual server.

Source: Direct LUN Access

39.4 Model



40 Data Access Component

Functionality to store and access data elements is provided by special components that isolate complexity of data access, enable additional data consistency, and ensure adjustability of handled data elements to meet different customer requirements.

40.1 Problem

How can the complexity of data storage due to access protocols and data consistency be hidden and isolated while ensuring data structure configurability?

40.2 Context

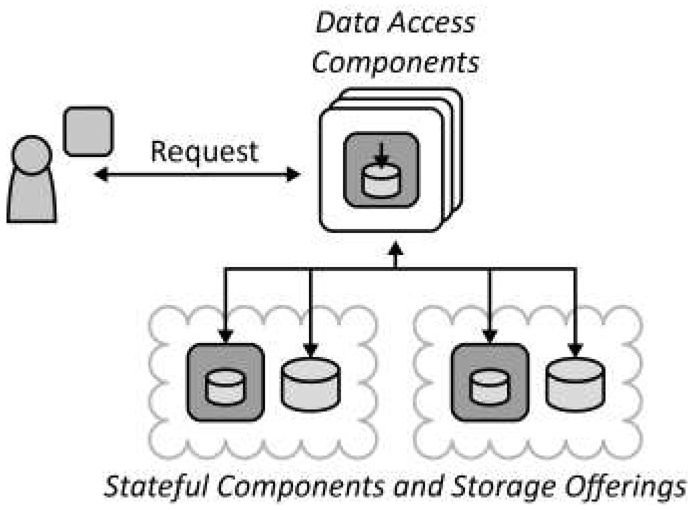
Handling the complexity of accessing data, i.e., handling of authorization, querying for data, failure handling etc. tightly couples application components to the used storage offering and complicates the implementation of these components as a lot of the idiosyncrasies of data handling have to be respected by them. Instead, different data sources should be integrated to provide a unified data access to other application components. Also, data may be stored at different cloud providers that have to be integrated as well.

40.3 Solution

Access to different data sources is integrated by a Data Access Component. This component coordinates all data manipulation. In case a storage offering shall be replaced or the interface of a storage offering changes, the Data Access Component is the only component that has to be adjusted.

Source: Data Access Component

40.4 Model



41 Automatically Defined Perimeter

In cloud architecture, IT boundaries are dynamic and can scale into multiple clouds from on-premise resources, which creates challenges when establishing and securing perimeters.

41.1 Problem

How can a perimeter be protected that is dynamic and extends from on-premise to multi-vendor cloud resources?

41.2 Context

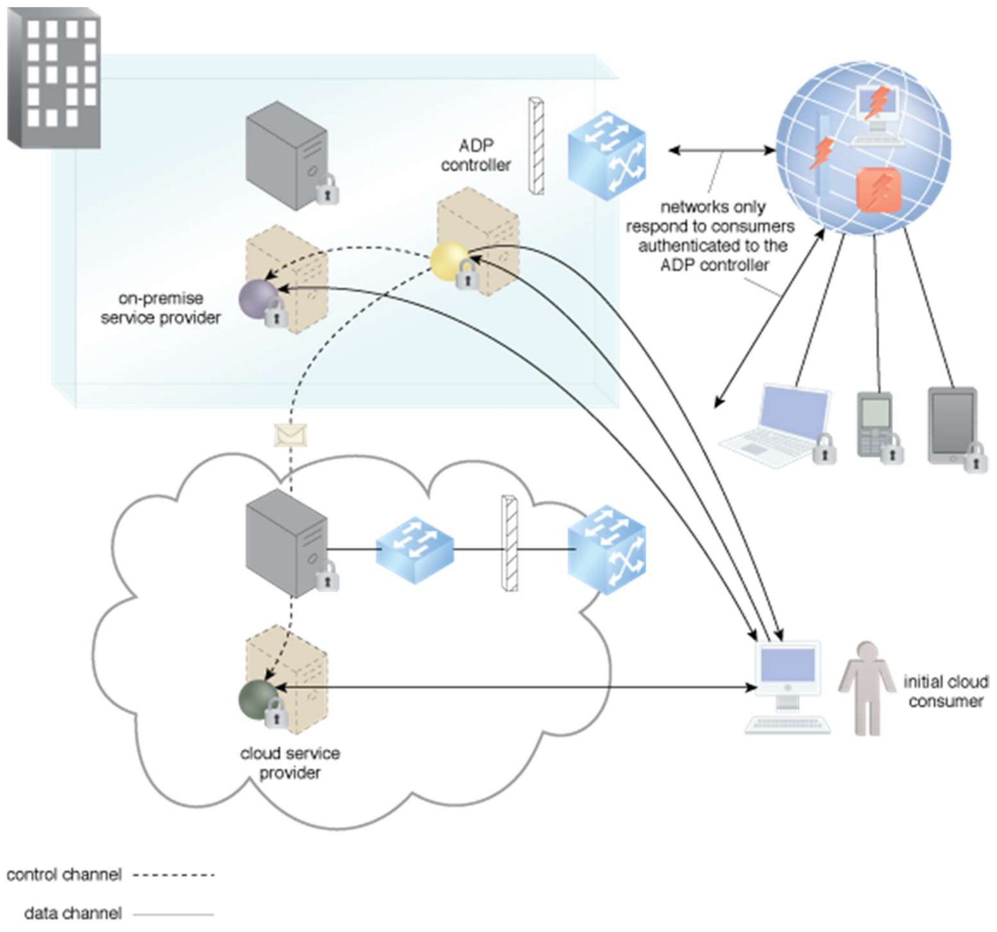
A system is established that provides protected communications between consumers and providers whereby each provider either accepts or rejects communications based on privileges securely granted automatically by a perimeter controller.

41.3 Solution

Cloud consumers authenticate to an automatically defined perimeter (ADP) controller which, if they are authorized, notifies the appropriate cloud provider services to respond to the authenticated consumer's requests. Otherwise, protected providers do not respond to any communications.

Source: Automatically Defined Perimeter

41.4 Model


ADP controller
networks only respond to consumers authenticated to the ADP controller
on-premise service provider
cloud service provider
initial cloud consumer
control channel
data channel

42 Distributed Application

Divide application functionality among multiple application components - each can then be scaled out independently (among different cloud providers).

42.1 Problem

Why to decompose application functionality into multiple, independent application components?

42.2 Context

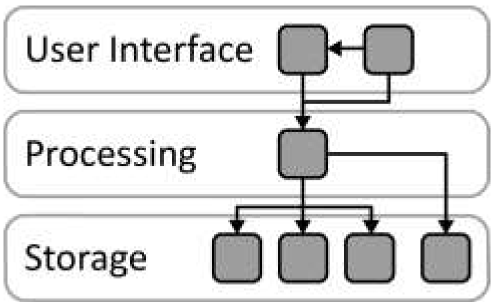
Cloud application architecture should make use of the cloud provider's distribution and scaling-out support system. The cloud application should employ multiple, possibly redundant IT resources for each component to ensure its availability.

42.3 Solution

The cloud application's functionality is divided into multiple, independent components - each providing a specific function. These components can be grouped together into tiers to denote that they should be deployed together physically, i.e., on the same server, cluster, or cloud provider.

Source: Distributed Application Pattern

42.4 Model



43 Hybrid Processing

Processing functionality that experiences varying workload is hosted in an elastic cloud while the remainder of an application resides in a static environment.

43.1 Problem

How can Processing Components that experiences varying workload be hosted in an elastic cloud while the remainder of an application is hosted in a static data center?

43.2 Context

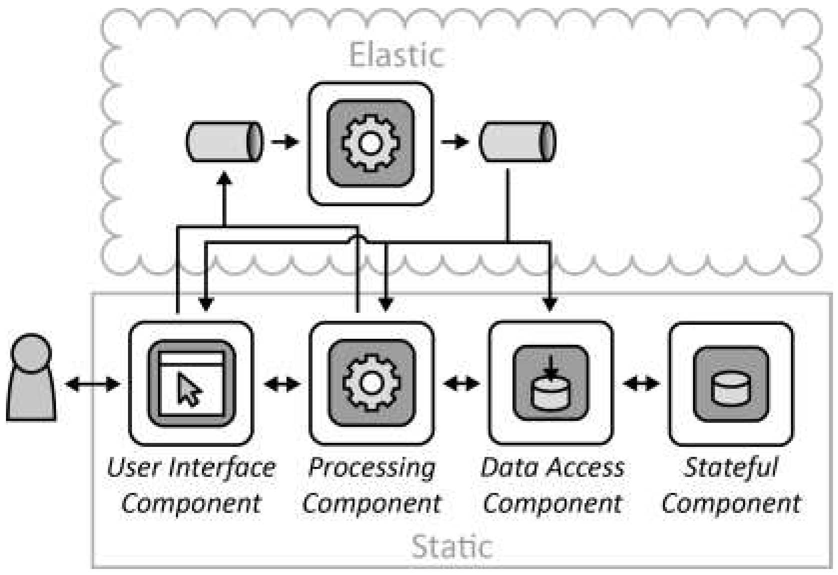
A Distributed Application provides processing functionality that experience different workload behavior. The user group accessing the application is, thus, predictable in size, but accesses the functions provided by the application differently. While most of the functions are used equally over time and, therefore, experience Static Workload, some Processing Components experience Periodic Workload, Unpredictable Workload, or Continuously Changing Workload.

43.3 Solution

The Processing Components experiencing varying workloads are provisioned in an elastic cloud. Loose Coupling is ensured by exchanging information between the hosting environments asynchronously via messages.

Source: Hybrid Processing

43.4 Model



44 User Interface Component

Interactive synchronous access to applications is provided to humans, while application-internal interaction is realized asynchronously when possible to ensure Loose Coupling. Furthermore, the user interface should be customizable to be used by different customers.

44.1 Problem

How can User Interface Components be accessed interactively by humans while being configurable and decoupled from the remaining application?

44.2 Context

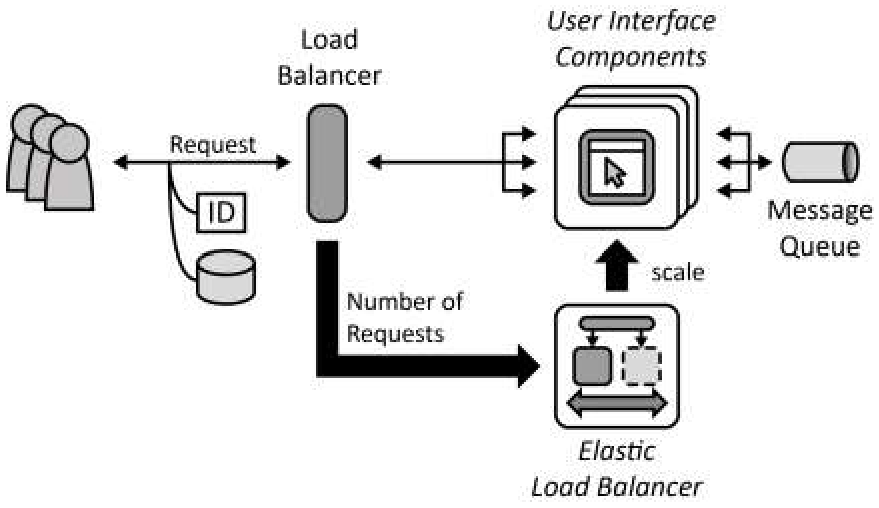
User Interface Component instances part of a Distributed Application have to be added and removed easily from the application without affecting the user experience. The dependencies on other application components should, therefore, be reduced as much as possible.

44.3 Solution

The User Interface Component serves as a bridge between the synchronous access of the human user and the asynchronous communication used with other application components. State information held externally, as described by the Stateless Component pattern. It is, therefore, attached to requests, may be held in a part of the user interface that is deployed on the user�s access device, or may be obtained from external storage. Instances of User Interface Components are scaled based on the number of synchronous requests to is as described by the Elastic Load Balancer pattern.

Source: User Interface Component

44.4 Model



45 Resiliency Management Process

Application components are checked for failures and replaced automatically without human intervention.

45.1 Problem

How can the overall availability of an application be ensured automatically even if individual application component instances fail?

45.2 Context

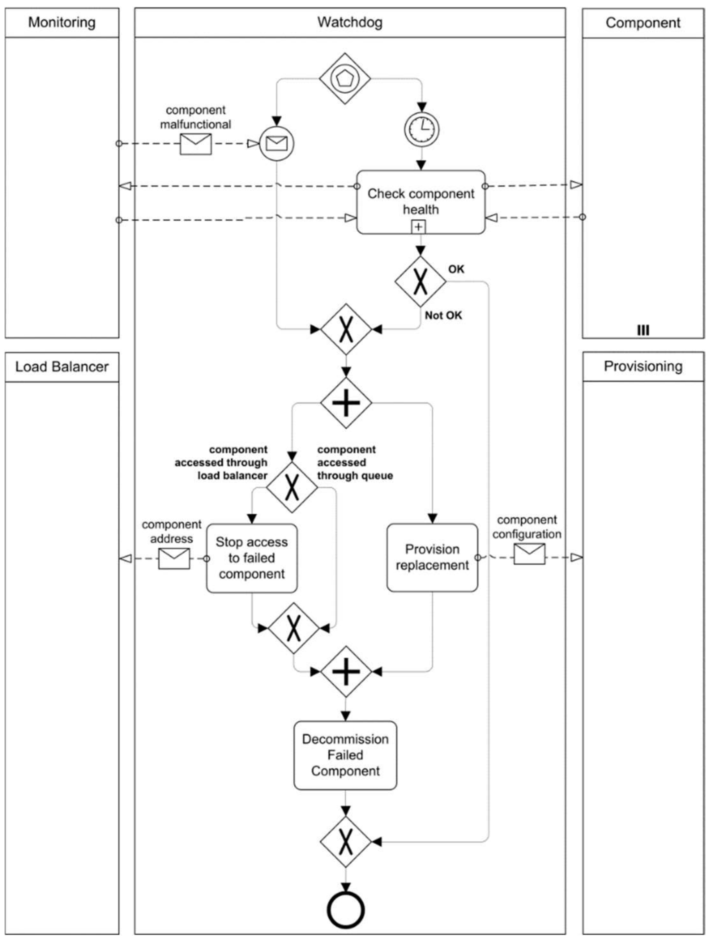
A Watchdog allows to monitor application components and react to failures. To handle this task, the component functionality must be verified and failing components must be replaced with newly provisioned components in a coordinated fashion.

45.3 Solution

This process is triggered by the monitoring functionality or by the Watchdog if it detects a component failure. Additionally, the Resiliency Management Process periodically verifies application component health. If a failure is detected, the faulty application component instance is decommissioned and replaced by a newly provisioned instance.

Source: [Provider Adapter]

45.4 Model



46 Leader Node Election

Sometimes the completion of a task requires the involvement of multiple instances of the same cloud service. If the service consumer invoking the cloud service instances does not have the necessary logic to coordinate them, runtime exceptions can occur leading to data corruption and failure to complete the task.

46.1 Problem

How can multiple instances of the same cloud service be coordinated to complete a greater task?

46.2 Context

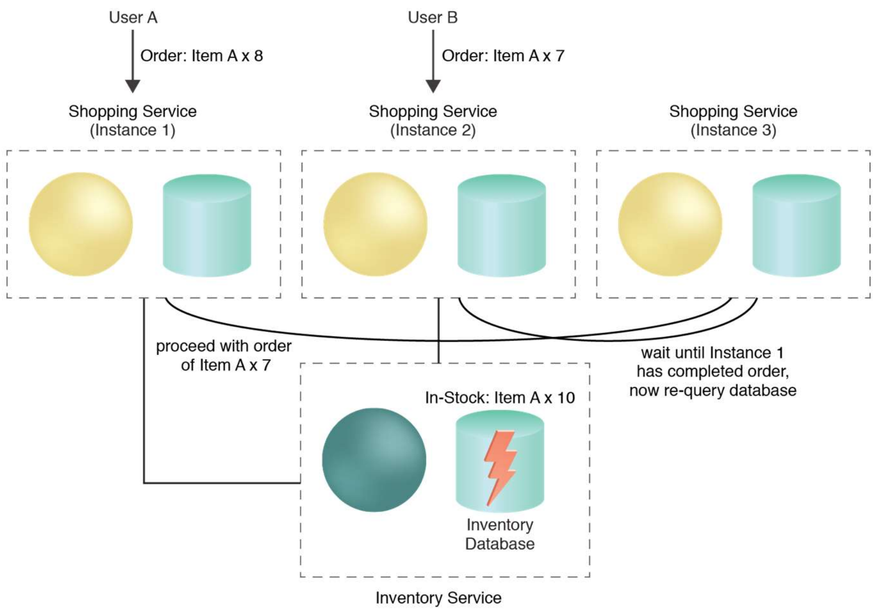
One of the invoked cloud service instances is designated as the leader node, responsible for aggregating the other cloud service instances in a coordinated effort to complete the task.

46.3 Solution

A reliable leader election mechanism is utilized, isolated from the cloud service runtime to elect the leader node.

Source: Leader Node Election

46.4 Model



User A
Order: Item A x 8
User B
Order: Item A x 7
Shopping Service (Instance 1)
Shopping Service (Instance 2)
Shopping Service (Instance 3)
proceed with order of Item A x 7
wait until Instance 1 has completed order, now re-query database
In-Stock: Item A x 10
Inventory Database
Inventory Service

47 Usage Monitoring

IT resources that are shared can generate a variety of runtime scenarios that, if not tracked and responded to, can cause numerous failure, performance, and security concerns and can further make usage-based reporting and billing impossible.

47.1 Problem

How can IT resource usage be measured?

47.2 Context

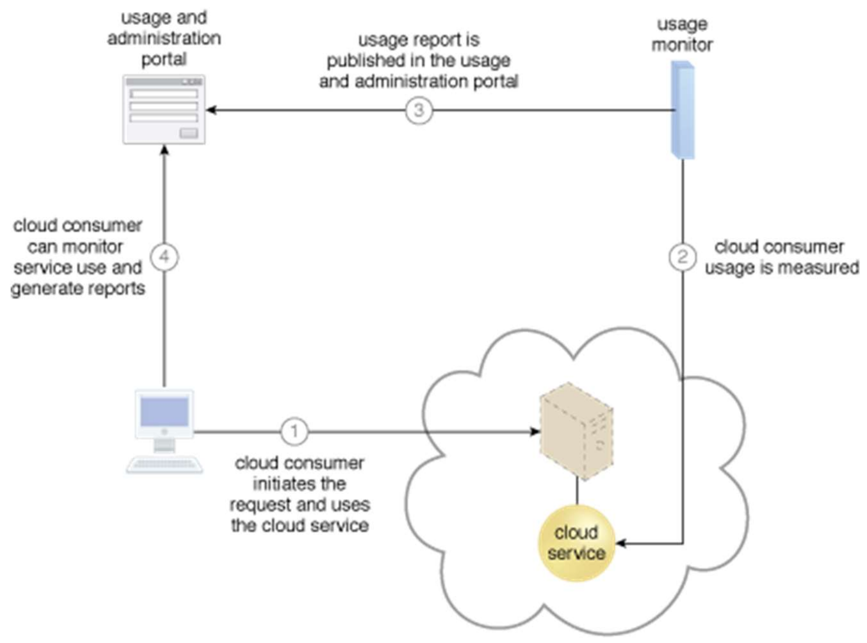
Cloud usage monitors are utilized to track and measure the quantity and nature of runtime IT resource usage activity.

47.3 Solution

Various specialized cloud usage monitors can be incorporated into a cloud architecture, most of which will interact with other IT resources to transfer or process collected usage data.

Source: Usage Monitoring

47.4 Model


usage and administration portal
usage report is published in the usage and administration portal
3
usage monitor
cloud consumer can monitor service use and generate reports
4
2
cloud consumer usage is measured
1
cloud consumer initiates the request and uses the cloud service
cloud service

48 Cross-Storage Device Vertical Tiering

Increasing the processing capacity of data stored on cloud storage devices generally requires the manual vertical scaling of the device, which is inefficient and potentially wasteful.

48.1 Problem

How can the vertical scaling of data processing be carried out dynamically?

48.2 Context

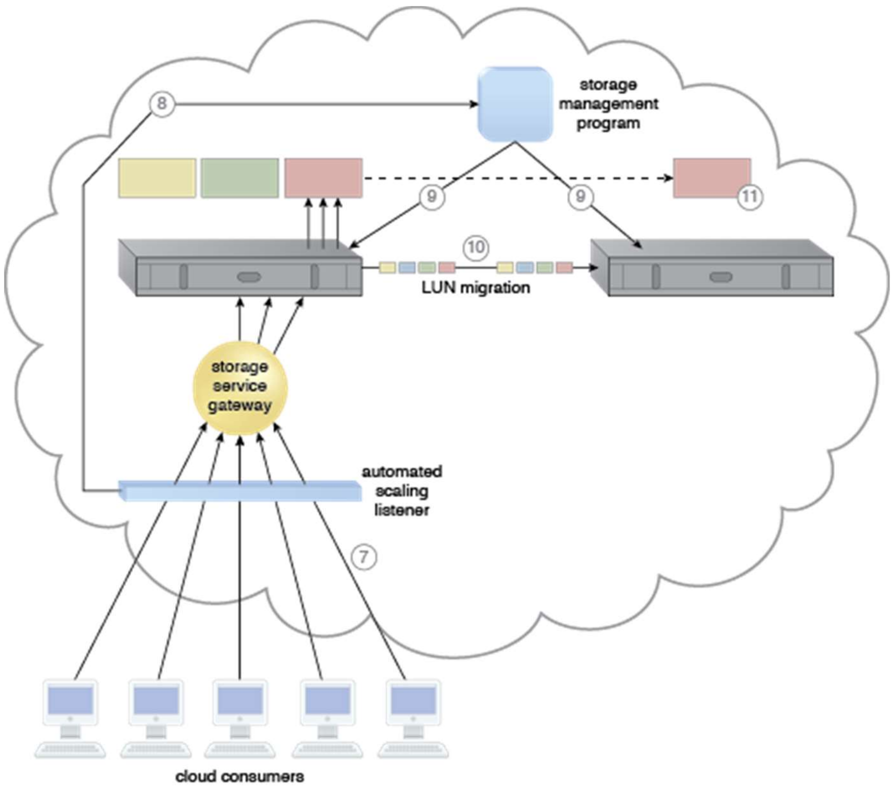
A system is established whereby the vertical scaling of data processing can be carried out dynamically across multiple cloud storage devices.

48.3 Solution

Using pre-defined capacity thresholds, LUN migration is used to dynamically move LUN disks between cloud storage devices with different capacities.

Source: Cross-Storage Device Vertical Tiering

48.4 Model


8
storage management program
9
9
11
10
LUN migration
storage service gateway
automated scaling listener
7
cloud consumers