Map workloads to the cloud

A guide to choosing cloud service and deployment models

Benedict Fernandes
Kyle Brown

September 12, 2016

The term "workload" has various meanings to people with different roles. These varied perspectives can lead to misunderstandings, miscommunication, and false expectations when planning to move or map workloads to the cloud or hybrid cloud. In this article, you learn what a workload is, understand the basic model and key attributes of a workload, and see how it fits in a cloud context.

Introduction

When teams adopt cloud technologies, they often blithely toss out the term workload in technical and project planning discussions. The problem is that this term means different things to people with different roles, leading to disagreement and improper expectations. This article explores these various perspectives. It provides guidance on how each of the perspectives, including business, architecture, user interaction, and data, among others, are all necessary to precisely define what a workload is. The first part of this article outlines a core and relevant set of attributes that qualitatively define a workload. The second part of this article addresses these considerations from a perspective of migrating workloads to the cloud and hybrid cloud.

A little history about the term

The term workload is not new in the field of information technology (IT). This term was coined in the early 1960s during the mainframe era. So, why are we attempting to discuss it again now, five decades later?

With the increasing adoption of cloud computing, it is difficult to have a conversation without referencing the term workload. For example, development teams might talk about migrating their workload to the cloud, struggling to move their workload to the cloud, running their workload in their private cloud, or changing and adapting their workload to the cloud. This term is generally overloaded with different meanings that often lead to ambiguity and miscommunication.

The term workload as commonly defined in computer science and the IT domain has primarily been characterized from a system resource usage or capacity perspective and, therefore, is very restrictive. On UNIX platforms, a workload was referred to in the context of the number
of processes that were running on the UNIX host. The UNIX operating system went as far as providing an uptime command. This command provided a convenient way for a system administrator to examine the workload that was currently running on the system and a means to assess whether the load on the system was increasing or decreasing.

Moving forward, as organizations build their cloud strategies and embrace associated technologies and practices, such as microservices and containers, a more inclusive and comprehensive definition of the term workload is warranted. Let's not forget, workload is a key asset that provides competitive value to an organization and is the primary reason why IT exists. Therefore, a more inclusive and comprehensive characterization of what constitutes the notion of workload, in this new evolving context of the cloud, is essential and is the objective of this article.

**Essential parts of a workload**

One fable that relates to our problem with the term workload is about seven blind men. Each man touches different parts of the body of an elephant and argues their interpretation of what the animal is from their limited perspective. The overarching lesson is that, although the individual perspectives were partially correct, the whole truth was an aggregate of these perspectives.

To understand what a workload (that is, the elephant in our analogy) is, you first need to understand the basic workload model and the services that build it.

**Workload model**

Whether you are a developer, tester, or administrator, for example, you often have visibility only to the part of a workload that applies to your role or rather your subset of the elephant. However, from a broader organizational perspective, other components constitute a workload from a business function, runtime execution, deployment, security, and management perspective. The following figure outlines the seven key attributes of a workload that are explained in the next section.
The application represents the core part of the workload. An application, typically consists of three components – a user interface, business logic (business and integration services), and data services. The application components implement the functional requirements of the workload. The deployment, security, and governance components typically represent the non-functional requirements of the workload.

Abstracting services behind APIs

A service is a fundamental building block of a workload. It is a stateless software component that can be independently packaged, deployed, secured, consumed in a self-service manner, and potentially managed in an automated way. The following diagram illustrates a schematic view of a service.

More broadly, a workload is a systematic collection of a set of inter-related services that meet both the intended functional requirements and the non-functional requirements. At the same time, a workload delivers on the overall proposed business value. At the core of this concept is the idea of abstracting business services and infrastructure services behind application programming interfaces (APIs).

Seven key attributes of a workload

As mentioned in the previous section, a workload has seven key attributes. Each one is explained here in more detail.
1. Strategic enterprise architecture

Each workload has strategic imperatives and business drivers. By understanding them, you can drive or validate many architectural and deployment decisions further downstream. The key task is to develop an enterprise architectural view of the workload at hand. This view provides insights into the key use cases, how clients and channels engage with the workload, the services that are offered, and how the services are orchestrated at a high level to fulfill the current business functions. The following figure illustrates this concept in a layered architecture.

As with any layered architecture, the strategic enterprise architecture has a separation of control. Each layer focuses on providing a specific function to the layer above it and consumes the services of the layer below it. The Channels layer represents the support for the various channels that are required by the enterprise. The API Services layer offers a programmatic entry point into the execution of the specific business process. The Business and Process Services require the business services to run any specific custom logic or transformation. The domain services typically represent the specific entity data that needs to be read or persisted to the enterprise data store. The persistence can happen to traditional database systems or to other enterprise systems by using a common and portable connector framework.

To acquire a complete understanding of a workload though, you must map how the various requests enter the system, how they are processed, and what external system interactions are necessary to service the request. Consider a typical example where a customer calls an agent in a call center. The call center agent enters the user’s information in a form that then starts a business process to retrieve the customer’s data from an external customer relationship management (CRM) system.

The enterprise architect generally provides or creates the strategic enterprise architecture. Then, the line-of-business (LOB) executives, application architeters, and other stakeholders use it to discuss the overall business and technical goals of the workload. They can use this all-encompassing architecture diagram to discuss both the business side and technical side of an organization.
The application architecture

The application represents the functional requirements of the workload and has three key constituents: a user interface, business logic, and data.

The business logic consists of common business services and integration services. This representation of an application is not new in IT. However, the way in which these parts are composed, exposed, distributed, and connected has changed over time with the innovations in the industry. Representational State Transfer (REST) is a common example of a key enabling technology that allows components in these various tiers to interoperate.

In the past, applications were packaged and deployed as a monolith and run on a single host, such as the mainframe. With the growth in PCs, client/server computing became attractive. It required a separation of the user interface, business logic, and data components. Then, multi-tiered computing evolved, with the emergence of the Internet, the web, and service-oriented architecture (SOA). And more recently, applications are adopting newer paradigms around microservices and use web-based technologies to rapidly develop applications. Figure 6 represents a generic four-tier distributed application architecture.

When you define an application, you need to understand the following characteristics:

- End-to-end application architecture and architectural style, for example: SOA, multi-tiered, or microservices
- Programming model, for example: Java Enterprise Edition (Java EE), Microsoft .NET, Spring, or Open Source MEAN stack
- How business processes and common services are exposed, for example: SOAP, REST, or sockets
- Protocols that are used for access and interoperability between the various tiers, for example: Remote Method Invocation (RMI), HTTP, TCP/IP, or Java Message Service (JMS)
- Enterprise integration resources and connectivity, for example: databases, SAP, mainframe, queuing, or CRM

We use the term polyglot to describe this diversity of characteristics for implementing the various components of an application.
2. UI services for the application

The user interface (UI) represents how the application interacts with the external world and has been the center of attention for well-known reasons. It dictates how input is received and how responses are rendered. The UI services should support the strategic enterprise architecture that was described in the previous section.

The application is channel-dependent, such as a browser, green screen, or mobile device. Therefore, you need to understand the following characteristics when you define an application:

- The UI component model
- The range of protocols that need to be supported based on the enterprise architecture
- The major frameworks that are employed
- How the UI services interact with back-end supporting services
- Whether the application supports APIs

Java EE applications typically use JavaServer Pages (JSP) or JavaServer Faces (JSF), where mobile channels are built with HTML5, JavaScript, and mobile frameworks. AngularJS front ends that integrate and use back-end services based on REST are becoming an increasingly common pattern.

For completeness, not all work is initiated by direct user interaction. An increasing number of businesses are exposing their services by using APIs. Batch applications for example, get triggered by a scheduler and require no user interface. B2B applications that integrate by using messaging is another class of applications that are event-based. They integrate programmatically and, therefore, do not need a user interface.

3. Business and integration services for the application

The business and integration services, also referred to as business logic, are the core part of the application. Key business processes are exposed as services to the channels and other consumers.

The key business processes are generally triggered from a user request from where it receives the necessary input to drive the business process. The business process can vary in complexity from a simple process to a very complex process that requires several back-end resources to complete execution.

When you examine the workload, you must understand the dependencies between the business processes and the supporting common services. The number of services and layers will vary from one application to another. You must analyze how these services are mapped from a logical architecture to a physical deployment topology. Also, you must comprehensively document the back-end enterprise resource dependencies that are needed for each scenario.

A typical example in this domain is a Business Process Execution Language (BPEL)-based process to file an insurance claim. This process potentially retrieves and stores information in a back-end mainframe system during its execution. The BPEL-process communicates with a SOAP-
based web service that, in turn, communicates with the mainframe by a service based on Java that is hosted on UNIX.

4. Data services for applications

As you observed previously, the business and integration services need to depend on the underlying data services to store and retrieve data. The underlying data store or domain model can vary based on how and where the data resides. Most often the data is persisted to a highly normalized relational database system. However, much of the enterprise data exists in hierarchical database systems on the mainframe. More recently, NoSQL data stores are becoming increasingly popular.

Therefore, it is common to have a polyglot persistence environment. In this environment, some data is in a relational system, such as Oracle, some data is on a mainframe CICS system, and some data is in a key-value store, such as IBM® WebSphere® eXtreme Scale® or Redis.

Other important considerations revolve around transactional integrity, confidentiality, partitioning, and latency. These considerations can get involved from a technical perspective but are essential considerations for completeness. Some use cases are a bit lenient on the data consistency requirements, where others have tight regulatory requirements about accuracy and reliability. Certain data models allow data partitioning to support high-levels of concurrency and eventual consistency to tolerate network failures. Data placement is another important consideration from a compliance, latency, security, and performance perspective.

The most popular data access pattern over the last decade is based on a Java Database Connectivity (JDBC) architecture. Data access services are generally built with persistence frameworks, such as Hibernate and Java Persistence API (JPA) architecture. Data services for mainframe, SAP, Siebel, or other similar enterprise systems leverage Java EE Connector Architecture (JCA) connectors to communicate with them.

A related consideration in data services is around accelerating application access by caching in a data grid to bring the data closer to where it is consumed. This approach provides several benefits mainly around faster response times, enhanced user experience, offloading back-end systems, and cost savings on the database infrastructure. WebSphere eXtreme Scale, Redis, Cassandra, and MongoDB are a few popular NoSQL data stores.

5. Deployment model

The deployment model is a software-to-hardware mapping. It represents a mapping from the application architecture to an appropriate production runtime architecture that is required to run the workload. This mapping is commonly referred to as a topology. The following figure illustrates a simple topology for an application server deployment.
This topology represents the application components, the supporting cast of infrastructure-related components, and their relative placement. Examples of infrastructure components include load balancers, proxy servers, firewalls, web servers, directories, and security servers.

Selecting the appropriate topology to satisfy the functional and non-functional requirements of the workload requires careful planning. Consider the following factors when you are deciding the correct topology:

- **Business imperatives.** Mission-critical workloads that require a high degree of isolation and control
- **High availability (HA).** A degree of component redundancy that is governed by HA requirements
- **Disaster recovery.** Meeting the Recovery Point Objective and Recovery Time Objectives
- **Security requirements.** The placement of security controls along various points of the request path
- **Performance.** The ability of the topology to meet the specified response times and throughput metrics
- **Scalability.** The ability to support growing workloads gracefully and as linearly as possible
- **Manageability.** The ability to provision, automate, and deploy the application and infrastructure

6. **Security model**

Considering the highly distributed nature of applications, especially mobile and Internet-based applications, you must clearly understand the security model of the workload. At a more granular level, the following elements collectively define the security model of a workload:

- **Authentication.** The mechanism to establish identity, for example: Is John really John?
- **Authorization.** Access control, for example: Is John authorized to perform action Y?
- **Confidentiality.** Data privacy, for example: Is John allowed to view data element Z?
- **Data integrity.** Ensuring that data was not modified by unauthorized users after it was sent
- **Non-repudiation.** The mechanism to prove that a user who performed an action cannot later deny it
- **Auditing.** Maintaining a record or log of security-level events of interest

The following figure illustrates these elements of a security model for Java workloads.
The specific security services that are used are often dependent on the underlying programming model and security services that are provided by the container. The following figure illustrates an example of the request flow that is augmented with the relevant set of Java EE security semantics.
The security aspects of the workload encompass more than the application layer. The requested and associated data typically flow through several intermediaries as shown in the deployment model in the previous figure. Therefore, from an end-to-end perspective, you must understand the security requirements. That is, you must specify the security configuration requirements all along the request path, from source to destination and back as shown in the previous figure. To understand the end-to-end semantics of the security model for the workload, you must also specify all security requirements for all intermediary components.

7. Governance model

Governance is about leadership and accountability. In the context of workload, it is the process to ensure a strategic alignment between the business goals and the management of the workload. Governance puts mechanisms in place to ensure that the workload is performing predictably and can reliably deliver on the goals of the business. Therefore, the governance process is responsible for instituting the following capabilities:

- Measure workload performance against key performance indicators.
- Track, enforce, and report on service level agreement (SLA) management.
- Aggregate, analyze, and synthesize various logs in the system.
- Provide a high availability plan that is based on business criteria.
- Articulate a disaster recovery plan and procedures to test the failover plans.
- Ensure that recovery point objectives (RPO) and recovery time objectives (RTO) are defined.
- Ensure that security standards are being met.

**Governance** is a good mix of product and process. It is necessary to use the right tool for the task. IBM offers a suite of IT Service Management products that you can use for application monitoring and SLA management. The high availability plan and disaster recovery plan are typically process documents that describe the procedures to follow when these events occur.

**Workloads in a cloud context**

Several factors go into making decisions around moving workloads to the cloud. Some of these factors are organizational, financial, regulatory, and technical in nature. For companies or startups
that do not have existing applications, the choice of computing platform is generally simpler. Cloud-based companies do not have to deal with migration and integration with older and often more closed systems. For them, moving or adapting to the cloud is relatively easier. These workloads are aptly referred to as "born in the cloud" or "cloud native applications."

Older enterprises do not have it that easy when migrating and integrating workloads to the cloud. They must first refactor or enable many of their existing workloads for the cloud. After they determine that the workload is cloud-ready, they need to make the following decisions:

- What is the most appropriate cloud service model?
- What is the most optimal cloud deployment model?

The following figure illustrates the two decisions that you need to make to map workloads to the cloud.

Selecting an appropriate cloud service model for a workload

The seven workload characteristics collectively help determine the target cloud service model. The following figure highlights the three cloud service models: software as a service (SaaS), platform as a service (PaaS), and infrastructure as a service (IaaS).
The following sections highlight the major factors that go into deciding between cloud service models.

**The IaaS model**

IaaS provides compute resources by using a self-service model. You might choose an IaaS model for the following reasons, among others:

- Your workload has stringent throughput and response time guarantees.
- Your workload is mission-critical and needs to manage the risk of availability. It also has a strict RPO and RTO.
- Your workload needs cloud benefits, such as capacity on-demand and rapid self-service provisioning. It also needs flexibility and control in terms of configuration management and performance tuning optimization.

**The PaaS model**

PaaS helps you develop applications in the cloud, for the cloud, and that are managed by the cloud. You might choose a PaaS model for the following reasons, among others:

- The business wants to get away from managing development, test, and runtime environments.
- The workload is an aggregation of primarily innovative cloud-based services. APIs are used heavily for internal integration and external partner integration.
- You can develop, run, and maintain the workload in the cloud by using platform services.
• The availability of a PaaS platform often depends on the availability and reliability of the underlying IaaS platform. When you deploy applications to a PaaS platform, you must be aware of these dependencies and their impact on the qualities of service, such as RPO, RTO, and the throughput and response times.

The SaaS model

SaaS provides purpose-built applications that can be consumed over the Internet. You might choose a SaaS model for the following reasons among others:

• Making the workload available as a SaaS offering is an up-front business decision: to make the software and function available as a subscription-based service that is accessible by using a browser, web APIs, or both.
• The provider of a SaaS offering has to use the underlying platform and build all the configuration elements into the software to ensure it is easily consumed, deployed, patched, managed, and available.
• You want to retain control over how it is configured, packaged, deployed, distributed, managed, and consumed.

Selecting an optimal cloud deployment model for your workload

You need to decide whether you will deploy your workload to a private cloud behind your corporate firewall or host it on a public cloud behind the firewall of the cloud service provider. To make this decision, you need to account for the key data points.

At a high level, the choice between private or public is guided by the requirements of the workload in terms of the level of control that is necessary and the degree of isolation or sharing that can be tolerated. Cost is definitely a consideration. However, when you make this critical decision, you need to account for the relevant organizational risks.

For example, your workload is deployed on a PaaS platform that is hosted on an IaaS-based infrastructure, such as Amazon Web Services. In this case, the overall availability and SLAs of your workload are strongly tied to the performance and availability of the underlying IaaS infrastructure. For high throughput, mission-critical applications, this model might not be acceptable.

Criteria for a private cloud

You can deploy your workload to a private cloud if the workload meets the following criteria:

• It is a high-volume, mission-critical workload that requires high levels of throughput and latency.
• It has regulatory requirements around security, privacy, confidentiality, and placement of data assets.
• Your environment has organizational IT maturity, skills, and hardware assets available that are necessary to host a private cloud.
• It needs deterministic RPO and RTO for high availability and disaster recovery.
• It requires customization and platform-specific optimization.
• It is an existing workload that is not easily enabled for public cloud consumption.

Workload examples that fit this model include a core banking system, hospital management system, stock trading website, high-volume commerce site, ticketing systems, or traffic control systems. You can also host these workloads in a public cloud. However, like all things, you need to account for risks in the planning.

Criteria for a public cloud

You can deploy your workload to a public cloud if the workload meets the following criteria:

• It uses cloud economics, such as the availability of utility pricing and lower up-front capital outlays.
• It is standardized with relatively large subscriptions.
• It is ad hoc, temporary, or seasonal, requiring dynamic just-in-time capacity provisioning.
• It can tolerate occasional latency and does not have strict SLAs for availability.

Workload examples that fit this model include test workloads, email systems, collaboration systems, storage workloads, analytics, or batch workloads. Netflix and Uber are widely used examples of public cloud. The Obama Presidential election campaign demonstrates an example of an ad hoc workload that was provisioned on the cloud, expanded the capacity manifold that led up to the election, and was then de-provisioned immediately after the election.

Hybrid cloud scenarios

More recently, the divide between a public and private cloud is getting increasingly gray. Most enterprises have existing IT assets and need to build on these investments as they embark on their cloud journey. Therefore, from a practical perspective, large enterprises are most likely to deploy hybrid cloud scenarios that require application and data components to be optimally placed in either in a private or public cloud. Cloud service and platform providers offer the necessary connectors, gateways, adapters, and so on to enable the various application and data integration hybrid cloud architectures.

To help you understand how a workload can run in a hybrid cloud architecture, we highlight three scenarios, which are illustrated in the following figure:

• **Scenario 1.** The presentation and business logic tiers run in a public cloud, and the data resides in a private cloud.
• **Scenario 2.** An application, typically a system-of-engagement, runs fully in the public cloud. A key characteristic of this scenario is that the relevant data to support this application is filtered and replicated from the private cloud to the public cloud in real time.
• **Scenario 3.** A cloud-based application aggregates data from back-end systems that are hosted in a private cloud and from data sources that are in the on-premises data center, such as a mainframe system-of-record.
At a fundamental level, a hybrid cloud implementation is a platform or ecosystem of capabilities. Collectively, they enable the workload to be distributed across heterogeneous environments, such as a public or private cloud or an existing traditional data center.

Conclusion

This article presented a comprehensive model to describe a workload. The model consists of seven key attributes that collectively articulate the architectural objectives and the functional and non-functional aspects of an enterprise application. This model is useful for migrating existing workloads to the cloud or the mapping of new workloads to a cloud-based architecture. These concepts provide a structure for promoting a more systematic discussion around architectural issues that are related to workloads in the emerging cloud context.
Related topics

- IBM Cloud Architecture Center
- developerWorks Cloud Computing hub: Cloud tools and techniques
- developerWorks Middleware library: Application development, cloud integration, and more
- IBM Middleware User Community
- Architecting the Cloud by Michael Kavis
- Hybrid Cloud for Dummies by Judith Hurwitz et. al.

© Copyright IBM Corporation 2016
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)