Managing application runtime environments in IBM PureApplication System

Bobby Woolf
Certified Consulting IT Specialist
IBM

April 2013
(First published 10 October 2012)

In IBM® PureApplication™ System, deployers install applications into runtime environments that administrators define by using cloud groups and environment profiles. As an administrator setting up a PureApplication System, what are the cloud groups and environment profiles that you will need to consider and create?

Introduction

In IBM PureApplication System W1500 v1.0 and W1700 v1.0 (hereafter called PureApplication System), deployers install applications (called pattern instances) into runtime environments that administrators define using cloud groups and environment profiles. As an administrator setting up a PureApplication System, what are the cloud groups and environment profiles that you need to consider and create?

For example, an application development lifecycle typically requires a separate runtime environment for each stage of development - these might be DEV, TEST, and PROD. These environments should be separated so that activities in each do not interfere with the others. Likewise, they may need further subdividing, such as sub-environments for multiple applications deployed to TEST or PROD or to distinguish different geographies or release levels. How can you use PureApplication System cloud groups and environment profiles to create separations like this?

This article is the third of three articles that explains the hardware and software foundation that PureApplication System provides for hosting application runtime environments:

• **Hardware**: A tour of the hardware in IBM PureApplication System explains the hardware that comprises of PureApplication System.

• **Virtualized hardware**: Best practices using infrastructure as a service in IBM PureApplication System explains how PureApplication System virtualizes its hardware to implement infrastructure as a service (IaaS).

• **Runtime environments**: The article that you are reading now explains how the virtualized hardware in PureApplication System is used to implement application runtime environments that workloads are deployed into.
Each article builds on its predecessor to explain this foundation fully.

To determine the cloud groups and environment profiles you will need, you need to first understand what these product features are for, which means first understanding some of the goals of cloud computing, and how hardware resources are virtualized. We will then discuss the features in PureApplication System for managing virtualized resources, and finally, a strategy for making use of these features. We will also consider what principles can be inferred from these strategies.

**Cloud concepts**

To understand PureApplication System, let's first review some of the basic concepts of cloud computing.

One goal of cloud computing is to optimize resource utilization, which can also be thought of as increasing application density. *Resource utilization* is the total amount of an available resource that is actually used, and the higher the better up to the resource's target limit. When usage of a resource is less than its target limit, the unused capacity and its associated costs is being wasted. Optimal resource utilization keeps resource usage as close to the target limit as often as possible. To optimize utilization, the system needs to manage each application's resource usage and predict that usage.

To better manage the applications in a cloud, a cloud computing system runs each application as a workload. A *cloud computing system* is a set of computer hardware and software that runs a cloud, specifically a cloud environment capable of running applications as workloads. A *workload* is a running program packaged as a virtualized application. A *virtualized application* is one that is installed independently of any particular set of hardware so that it can run anywhere on a virtualized platform. A *virtualized platform* is a set of hardware configured with one or more hypervisors to run its software as virtual machines. The platform may also virtualize the access to storage and networking resources. A *hypervisor*, also know as a virtualization manager or platform virtualizer, is a specialized operating system that only runs virtual machines. A *virtual machine* (VM) is software that simulates a physical computer by running an operating system just like a computer, but the computer is a virtual one because the VM's hypervisor decouples the operating system from the underlying hardware. This approach enables the hypervisor to run multiple operating system instances on a single set of hardware (such as a single physical computer), each as a virtual machine, and to manage each one's access to the hardware resources. It also makes each VM highly portable so that it can run anywhere on the virtualized platform.

*Navigating the IBM cloud, Part 1: A primer on cloud technologies* explains platform virtualization using hypervisors. It shows the virtualization stack shown in Figure1, virtual machines running in a hypervisor that runs either directly on the physical hardware (Type 1), or in a host operating system on that hardware (Type 2). Each VM requires its own unique IP address so that it can participate on the network as a standalone computer. The hypervisor itself requires another IP address so that it can be managed remotely.
Resource utilization

A cloud computing system provides its workloads access to virtualized computing resources as needed. The system manages the resources each workload is allowed to use to ensure that all of the workloads run successfully.

To manage the workloads' access to the virtualized resources, the system employs multiple competing approaches:

- **Isolation**: Workloads can be isolated from each other so that problems in one workload (such as a runaway process that consumes all CPU or memory) do not affect the others.
- **Sharing**: Workloads should draw from common pools of resources so that some may use more resources when others do not need them.
- **Allocation**: Each workload or set of related workloads should get a bounded set of shared resources to ensure it gets the minimum it needs to run successfully, but limits its growth so that it does not take more than its fair share.

These competing approaches must be balanced. If all workloads are completely isolated, that is the old computing model where each application is deployed on separate hardware. If workloads share all resources, whichever ones take the most resources first can starve the remaining workloads. Allocation strikes a balance by grouping workloads, allowing sharing within a group by allowing each group to draw from pools of common resources, yet making sure that each group gets a bounded amount of resources. This helps ensure that every workload gets its fair share of resources.

The goal of this combination of approaches is to optimize resource utilization. Isolating resources or assigning dedicated resources to a workload lowers utilization by forcing the capacity to remain unused when the workload does not need it. Conversely, resource sharing helps alleviate this problem and increases utilization by allowing one workload to use a resource when another does not need it.
Let us explore these approaches in greater depth and see how they help optimize resource utilization.

### Resource isolation

*Resource isolation* creates figurative walls between sets of resources so they can operate independently. That way, problems in one walled area do not affect the other walled areas. A consequence is that resources cannot be shared across walled areas.

Two main aspects of isolation in cloud computing are:

- **Computational isolation**: Groups of CPU and memory capacity are separated from each other. When two workloads execute with computational isolation, one workload's consumption of those resources does not affect the other workload. The isolation can be physical or virtual.
  - **Physical computational isolation**: Also called dedicated resources, this means that each group of resources is composed of separate chips on the circuit board.
  - **Virtual computational isolation**: Also called virtualized resources, this means that a virtualization layer creates groups of seemingly separate resources that may actually share the same chips. The isolation provided is only as good as the virtualizer (typically a hypervisor) that implements it.

- **Network isolation**: Communication flows between computational resources via separate connections. The isolation can be physical or logical.
  - **Physical network isolation**: This means that separate network connections run on parallel sets of network equipment (such as network interface cards (NICs), cables, switches, and so on). This way, the signals for one network never transmit on the hardware for the other networks.
  - **Logical network isolation**: This means that separate network connections run on the same network equipment, so they share the same bandwidth, but their packets are routed through the shared hardware separately via different broadcast domains. A broadcast domain is typically implemented as a virtual local area network (VLAN).

### Resource sharing

*Resource sharing*, also known as resource pooling, enables multiple workloads to access their resources from pools of shared resources. The workload does not care which resource it gets from a pool; they are all equivalent and interchangeable. When a workload needs a significant amount of resources, it takes multiple items from the pool; at other times, it takes fewer items from the pool. When it is finished using a resource, the workload releases it back to the pool.

The advantage of sharing is that the system can assign resources dynamically, shifting over time from the workloads that need fewer resources to those which need more. A consequence of sharing is that misbehaving workloads can consume too many of the resources in a pool, thereby starving the other workloads.

### Resource allocation

*Resource allocation*, also known as logical isolation, is an approach to set boundaries on a workload by putting lower and upper limits on resource sharing. Allocation ensures that a workload
gets at least the minimum resources it needs and cannot consume more than its fair share of a pooled resource. The system can set allocation limits on any shared resource: CPU, memory, storage, bandwidth, even software licenses. For example, perhaps two workloads sharing a pool each specify CPU usage at 5-10 CPUs. With this setting, each workload is guaranteed that at least 5 CPUs will be available, yet when the workload grows, it does not get more than 10 CPUs.

Allocation balances isolation and sharing, finding a happy medium between every workload getting its own dedicated resources but no sharing and uncontrolled sharing that allows any one workload to grab all of a shared resource. Allocation enables sharing but within limits.

**Optimized utilization**

Resource sharing helps increase utilization, but resource allocation is still needed to make sure the resources are shared fairly. This then raises the question of how allocations should be set. The safest approach to allocating a shared resource is to promise workloads a total amount that is no greater than the target limit for that resource. Then, even if every workload simultaneously demands its full allocation, each workload receives the resources it requests and the resource is fully utilized. Yet most workloads use their full allocation only some of the time. If the allocation is based on average demand, then the workload is starved for resources when its load is above average, and it leaves capacity unused when its demand is below average. If the allocation is based on peak demand, the workload always gets the resources its load requires, but even more capacity remains unused more of the time. Therefore, allocating to the workloads only the resource capacity available leads to only partial resource utilization, not the full utilization desired.

Figure 2, taken from *Navigating the IBM cloud, Part 1: A primer on cloud technologies*, shows what resource allocation based on peak demand looks like. Because resource utilization at any particular time is at most equal to system capacity and usually much less, this means that much of the resource capacity remains unused, lowering resource utilization.

**Figure 2. Resource utilization**

Resource allocation can be leveraged to increase utilization. *Resource over allocation* is a technique where the system intentionally promises more capacity than it actually possesses. That is, enough of the workloads intentionally underutilize enough of their allocations so that the system does have sufficient capacity to meet actual demand. This technique is common for allocating physical
assets with high opportunity costs: airlines sell more seats than the airplane contains, hotels allow reservations for more rooms than exist in the building, and banks store much less cash in their vaults than the sum of their customers' account balances. Over allocation is not necessarily a bad practice, but rather is a wager that actual demands at any specific time by a diverse group of consumers will be less than what the individuals collectively anticipate.

Resource over allocation is not actually a problem, only a potential problem. It creates the opportunity for resource contention, which is an actual problem. Resource contention occurs when demand for a shared resource exceeds the system's capacity of that resource. Too many workloads expect too much of the resources they were promised, collectively more than the system has available. In the physical world, this manifests itself as an airline with an overbooked flight, hotel guests having to share rooms or be accommodated in other properties, and a financial institution facing a bank run. Resource contention is a major reason why nations go to war. Hopefully, cloud computing can help prevent workloads and their stakeholders from going to war with each other.

Figure 3 shows what over allocation, and specifically, resource contention look like. The system has a certain capacity for a resource, but has promised greater capacity to the workloads that result in over allocated capacity. As long as the workloads demand less total resources than the capacity available, everything is fine. But, when the workloads demand too much of what the system has allocated to all of them, then total demand is greater than the resources available and resource contention occurs. The system must resolve it to keep the actual usage below capacity.

Figure 3. Resource contention

The system must resolve resource contention when it occurs. By default, hardware tends to resolve resource contention by crashing - either individual processes or the entire operating system stops running. Simplistic efforts to avoid crashing are often not much better. For example, the operating system may sacrifice the single process that is consuming the most resources, but that is probably the one supporting the most users, and therefore, the most important one to preserve! Intelligence is needed to resolve resource contention satisfactorily. Manual human intervention is usually unsatisfactory because contention needs to be resolved immediately and cannot wait on a committee to determine a course of corrective action, much less wait for the action to be applied. The intelligence to resolve resource contention must be automated so that it can be applied quickly.
Resource contention management is automated intelligence to resolve resource contention. The management must pick winners and losers among the workloads competing for the over allocated resource, decision making based on a combination of workload prioritization and resource rationing.

- **Prioritization**: This technique assigns resources to workloads by importance. A workload’s priority may be predetermined by status, or the system may assign priority dynamically such as on a first come, first serve basis. However priority is assigned, to enforce it, the system fulfills the requests of the highest priority workloads first, and continues to do so for workloads of progressively lower importance until it runs out of shared resources, at which point it rejects the requests of the remaining lower priority workloads. The consequence of prioritization is that the higher priority workloads do not suffer at all but the lower priority ones suffer greatly.

- **Rationing**: This technique gives each workload only a portion of the resource it requests. Typically, the portion is the ratio of available resources to total requested resources. Thus bigger workloads receive a greater absolute share of resources, but also suffer a greater absolute shortfall. The consequence of rationing is that all workloads get part of what they request, but are all partially starved for the constrained resource.

To summarize, cloud computing employs techniques to optimize resource utilization. One technique, resource over allocation, increases resource utilization, but it creates the opportunity for resource contention. When resource contention occurs, the system must use resource contention management to resolve it though a combination of prioritization of workloads and rationing of the resource. Increasing over allocation usually increases utilization, but it also increases the likelihood and severity of resource contention occurrences.

**PureApplication System environment features**

To understand how best to use PureApplication System for isolating workloads while sharing resources, it is helpful to first understand the main features the product provides for sharing resources as a cloud.

Five types of system resources in PureApplication System are involved in deploying and running pattern instances. These define the runtime environments available for a pattern to be deployed to and control the resources available for a pattern instance to run in. They are:

- **Compute node**: This is a set of computer hardware containing CPU and memory that has access to storage and networking.
- **IP group**: This is a set of IP addresses, the ID of the VLAN they will use to communicate, and settings for how to connect to the network the VLAN is part of.
- **Cloud group**: This is a collection of one or more compute nodes and one or more IP groups. It is essentially a logical computer. It physically isolates resources.
- **Environment profile**: This is a policy for deploying patterns into cloud groups. It creates logical isolation of resources by allocating the resources.
- **User group**: This is a list of users in the same role, a role that can use environment profiles to deploy patterns.
An environment profile associates user groups with cloud groups so that the users in those user groups can deploy patterns to those cloud groups, as shown in Figure 4. An environment profile grants access to user groups to specify who can use the profile to deploy patterns. A profile can grant access to multiple user groups and a user group can be granted access to multiple profiles. An environment profile also specifies what cloud groups it can deploy to. Multiple environment profiles can deploy to the same cloud group, and a profile can deploy to multiple cloud groups.

**Figure 4. Relationship of PureApplication System resources**

![Diagram showing relationship of PureApplication System resources](image)

Figure 5 shows what a few instances of these system resources might typically look like.

**Figure 5. Typical PureApplication System resource instances**

![Diagram showing typical instances of PureApplication System resources](image)

For a given combination of a user group, an environment profile, and a cloud group:

- The user group specifies which users can use the environment profile to deploy patterns.
- The environment profile specifies that the users can deploy patterns to the cloud group.
- The cloud group specifies the hardware (specifically, compute nodes and IP groups) the deployed patterns will run on.

The settings in an environment profile control the resources in a cloud group that are assigned to a pattern instance as it is deployed. These become settings in the pattern instance that control how the cloud group runs and manages the instance. This helps coordinate and control how multiple pattern instances are deployed and run in the same cloud group. If the instances are deployed via different environment profiles, the instances' settings can be set differently to use different resources with different limits. The **Environment profile** section has more details.

The PureApplication System administration console is divided into two main tabs:
• **System Console:** This is for the admin role. It provides access to artifacts for operations, intended to be managed by hardware and cloud administrators, in other words users who manage the PureApplication System.

• **Workload Console:** This is for the deployer role. It provides access to artifacts for development, intended to be managed by workload administrators, such as users who develop and deploy patterns on the PureApplication System.

Cloud groups are managed on the system console whereas environment profiles are managed on the workload console. Pattern deployers should think of environment profiles, not cloud groups, as their deployment targets. Nevertheless, focusing on environment profiles as a way to hide cloud groups is a leaky abstraction – when you deploy a pattern, you choose an environment profile. However, for each virtual part, you then need to choose a cloud group, or at least accept the default.

**Compute node**

A *compute node* is essentially a very compact computer, rather like a blade server. As a computer, it is possible for a compute node to run an operating system. However, in PureApplication System, instead of a traditional operating system, each compute node runs a hypervisor.

Earlier, Figure 1 showed how a hypervisor enables multiple virtual machines to share an underlying set of hardware. In PureApplication System, as shown in Figure 6, the physical hardware is a compute node, the Type 1 hypervisor runs directly on the compute node, and each virtual machine runs a middleware server in an operating system. Technically, the VM can run any programs that can be installed in the OS. To deploy that VM on PureApplication System, it will need to be developed into a virtual appliance, as discussed in the article *Navigating the IBM cloud, Part 1: A primer on cloud technologies*. Nevertheless, PureApplication System uses its VMs primarily to run middleware servers.

**Figure 6. Hypervisor stack in a compute node**

Here is a brief summary of the hardware in a compute node. If you are curious about more hardware details, see *A tour of the hardware in IBM PureApplication System*. 
A compute node consists of:

- **CPU**: An Intel® compute node contains 16 physical (32 logical) cores.
- **Memory**: An Intel compute node contains 256 GB of RAM.
- **Storage**: A compute node contains an 8 GB SAN adapter.
- **Networking**: A compute node contains a 10 GB Ethernet adapter.

It has access to resources shared by all compute nodes:

- **Storage**: A PureApplication System includes two IBM Storwize V7000 storage units with 6.4 TB SSD and 48 TB HDD of storage accessed as a SAN.
- **Networking**: A PureApplication System includes two BLADE Network Technologies (BNT) 64-port Ethernet switches that, together, form the hub of its internal networking hardware and enable it to connect to the enterprise's network.

**IP group**

An **IP group** is a set of IP addresses, expressed either as a list or range. Since a network does not work properly with duplicate IP addresses, each address in the group is unique and an address can only belong to one group. The group also has settings for the ID of the VLAN they should use for communication, and settings for how to connect to the external network the VLAN is part of. All of the addresses in the set must belong to the same subnet of the external network, the one indicated by the netmask.

IP groups serve three functions:

- **Dynamic IP addresses sharing**: An IP group is a shared pool of addresses that the system draws from when deploying the VMs that compose each pattern instance.
- **IP address pool isolation**: Two teams can each use a different group so that if one team deploys too many applications and consumes all of its addresses, the other team can still deploy applications because it is using a separate pool.
- **Logical network isolation**: The VLAN specified by the IP group's VLAN ID enables the group's applications to communicate on an isolated logical network.

A **virtual local area network** (VLAN) logically isolates its network traffic from that of other VLANs on the same network as a separate broadcast domain. This means that the network traffic of applications using two IP groups with different VLAN IDs transmits on (seemingly) independent networks. This is helpful, for example, to deploy a pattern's HTTP servers on a separate network from the WebSphere Application Server custom nodes so that a network firewall can be placed between those two tiers. It is also helpful to isolate the network traffic of unrelated applications (like development vs. production, or the finance department vs. the HR department).

PureApplication System does not define VLANs - those are defined on the network by the network administrator - but, it does make use of VLANs extensively. PureApplication System uses VLANs in two distinct ways, either as a management VLAN or an application VLAN.

A **management VLAN** is used internally by the system.
• **Communication**: These VLANs enable the system's internal processes to communicate with each other.

• **IP addresses**: The system uses its own internal IP addresses, so the network administrator does not need to provide any IP addresses. The network administrator still needs to reserve each VLAN ID on the network so that the network does not create another VLAN with the same ID.

• **Scope**: Traffic on these VLANs only flows internally on the system. It should never flow on the enterprise network externally from the system.

An application VLAN is used by business applications deployed onto the system.

• **Communication**: The VLAN enables the applications to communicate within themselves, with each other, and with resources on the network.

• **IP addresses**: The network administrator must provide not only the VLAN ID for each of these VLANs, but also a pool of IP addresses that the applications use to connect to the network. The network administrator should reserve the VLAN ID and IP addresses so that the network does not use these values elsewhere.

• **Scope**: Traffic on each of these VLANs flows internally on the system and externally on any parts of the enterprise network configured to also use the VLAN.

PureApplication System itself requires three management VLANs, plus each cloud group requires another management VLAN. Each IP group requires an application VLAN. While all of the IP groups can share the same VLAN, typically different groups of applications use separate VLANs. The appropriate combination of IP groups, application VLANs, and sets of IP addresses depends on how your enterprise architects and network administrators want to isolate your applications' network traffic.

IP address assignment from an IP group follows a lifecycle typical of a shared resource composed of discrete units:

• The system assigns an IP address to a VM when its pattern is deployed (that is, the assignment occurs when the VM is created as part of its pattern instance being created). Similar to a DHCP server, an IP group supplies IP addresses when the deployment process requests them.

• That address remains assigned to the VM even when the pattern instance is stopped or stored.

• The address is returned to the IP group when the pattern instance is deleted.

To define a group, the enterprise's network administrators specify all of the group's settings: the set of IP addresses, the VLAN ID, and the other settings that specify how to connect to the enterprise network like gateway, subnet, and DNS host. The PureApplication System administrator does not get to choose these settings. He uses the settings given to him by the network administrator to define the IP group.

For a given set of network settings from the network administrator, the PureApplication System administrator gets to choose how many IP groups to set up. What the network administrator specifies is:
• A set of network settings: gateway, subnet, DNS, and so on
• The ID for a VLAN on that network
• A set of IP address on that VLAN

In turn, the PureApplication System administrator captures these settings as one or more IP groups. Each of the IP groups will have the same network settings and VLAN ID. The difference is that the set of IP address can be split across multiple IP groups. An IP address can only belong to one IP group, so the number of IP groups possible is as small as one and as large as the number of addresses in the set. For example, if a set contains ten IP addresses, you can create any one of these options:

• One IP group with all ten addresses
• Two IP groups with five addresses each
• Ten IP groups with one address each

Again, each of these groups has the same VLAN ID and other network settings. All that varies is the set of IP addresses.

Cloud group

A cloud group is a virtualized platform for running workloads, and acts like a logical computer. It accomplishes two main goals:

1. **System segmentation**: It divides a PureApplication System into one or more logical computers. Groups run isolated from each other.
2. **Compute node aggregation**: It groups one or more compute nodes along with at least one IP group into a logical computer that can have greater capacity than a single node.

**Figure 4** shows the relationship between cloud group, compute node, and IP group. A cloud group contains compute nodes and IP groups.

A cloud group, in addition to containing a set of compute nodes and IP groups, has three main properties:

• **Name**: This is what you call the cloud group; for example: DEV, TEST, or PROD.
• **Type**: This setting can be thought of as the resource over allocation policy for the cloud group. It defines how resources, specifically CPUs, are allocated to virtual machines (VMs) during pattern deployment. It affects how much CPU capacity is available to a VM and to all of the VMs deployed with an environment profile, which is especially relevant at times when user load is high on multiple applications.
• **Management VLAN ID**: This is the ID for a VLAN that the cloud group uses to enable internal communication between its VMs. It must not already be in use by the network. The VLAN does not require any IP addresses because the VMs are assigned addresses from the IP groups.

The type setting for a cloud group has one of two possible values: dedicated or average.
• **Dedicated**: This is no CPU over allocation. It is appropriate for applications whose usual state is a high user load, such as production applications. For a cloud group with this policy:
  - 1 virtual CPU = 1 physical CPU
  - The 16 cores on an Intel compute node are allocated as 16 CPUs.
  - VMs of the patterns deployed to this cloud group are expected to highly utilize the CPU capacity assigned to them, which is typical of applications that frequently experience high user load.
  - Fewer patterns can be deployed to a cloud group with this setting, but they avoid resource contention caused by this method of CPU over allocation.

• **Average**: This is a 4-to-1 CPU over allocation. It is appropriate for applications whose usual state is a low user load, such as development applications. For a cloud group with this policy:
  - 4 virtual CPUs = 1 physical CPU
  - The 16 cores on an Intel compute node are allocated as 64 CPUs.
  - VMs of the patterns deployed to this cloud group are expected to under utilize the CPU capacity assigned to them, which is typical of applications that need to be available, but are not used heavily.
  - You can deploy more patterns to a cloud group with this setting. However, their CPU over allocation means that when the applications are used heavily, they encounter resource contention that leads to degraded performance.

A cloud group acts like one big logical computer that is a virtualized platform. Each virtual machine that runs in a cloud group executes in one of the cloud group's compute nodes and runs with an IP address from one of the IP groups.

A particular compute node can belong to only one cloud group (at most). Typically, a cloud group contains at least two compute nodes so that the group can keep running even if one of the nodes fails. This limits the number of cloud groups that a single PureApplication System can support. For example, a small Intel configuration has six compute nodes, which means that it can support a maximum of six cloud groups, and probably supports just three cloud groups (assuming two compute nodes per group). A particular IP group can belong to only one cloud group (at most), so a lack of IP groups can limit the number of cloud groups that can be created. Each cloud group also requires its own management VLAN, so a lack of VLANs assigned by the network administrators can limit the number of cloud groups that can be created.

Cloud groups create isolated runtime environments, such that workloads running in one group are not affected by workloads running in another group.

**Environment profile**

An environment profile defines policies for how patterns are deployed to one or more cloud groups and how their instances run in the cloud group. To deploy a pattern, a user selects a profile for performing the deployment, which in turn specifies the cloud groups the deployer can deploy patterns to. The deployer should think of the environment profile as the target he is deploying a pattern to. The fact that the profile deploys the pattern instance into a cloud group is a system-level detail that a workload-level user, like a deployer, does not need to be aware of.

The profile specifies several configuration settings:
• **Access granted to:** This is who is allowed to use the profile to deploy patterns.

• **Deploy to cloud groups:** This is the list of cloud groups this profile can deploy patterns to, and for each cloud group, which IP groups the deployer can use.

• **IP addresses provided by:** This is how the deployment process assigns IP address from the group.
  - **IP Groups:** This means automatic, that addresses are selected by the deployment process.
  - **Pattern Deployer:** This means manual, that addresses are selected by the deployer during the deployment process.

• **Environment limits:** This enforces limits on the resources available to pattern instances that are deployed through this profile.
  - **Computational resources:** These are resources such as CPU, memory, and storage.
  - **Licenses:** This is PVUs per product

• **Deployment priority:** This is used to prioritize pattern instances dealing with resource contention (deployment, runtime resources, and failover). See the PureApplication System virtual machine behavior section. The possible priority levels are Platinum, Golden, Silver, and Bronze.

• **Virtual machine name format:** This is a naming convention used when creating virtual machines.

• **Environment:** This is a list of environment roles, which is a label for the users' convenience that is used in the pattern deployment properties dialog to filter profiles (see the Type field in Figure 7). The possible roles are Production, Test, and so on.

Multiple profiles can deploy to the same cloud group, and a single profile can deploy to multiple cloud groups (see Figure 4 and Figure 5). Not all deployers have access to all of the system's environment profiles (see User group below). To deploy to a particular cloud group, a deployer needs access to an environment profile that can deploy to that cloud group. That profile will in turn set policies about how those patterns are deployed, such as which IP groups are made available for use and setting the resource limits.

Different profiles enable users deploying potentially the same pattern to the same cloud group to do so with different limits and different settings applied to the pattern instances. When an environment profile deploys a pattern instance, it allocates a portion of the cloud group's shared resources to the instance, creating logical isolation of the instance. Profiles can place limitations on separate teams deploying to the same cloud group to logically isolate their pattern instances and prevent them from using too many resources, such as using the same IP group (and therefore, all of the IP addresses), or too much of the underlying CPU capacity. A profile also sets properties of a pattern instance that are enforced when the cloud group runs the instance.

When two user groups share an environment profile (that is, both are assigned to it), this means that both have the same pattern deployment policies and their pattern instances share the same resource allocations. For two user groups to have different policies or separate allocations of resources, each group needs to be assigned to its own environment profile. Two groups sharing the same profile means that both groups' deployments count against the same limits. Two groups using different profiles means that each group’s deployments count against its own limits.
User group

A user group represents a role, a set of types of tasks to be performed and the permissions need to perform these task types. A user group has two main properties:

- **Group members**: A list of users who perform this role.
- **Permissions**: Capabilities needed by users in this role to perform their tasks.

One of the main functions of a user group is to specify which users can use a particular environment profile to deploy a pattern.

PureApplication System virtual machine behavior

An application is deployed to PureApplication System as a pattern instance composed of virtual machines for running the middleware the application runs in. The behavior of these virtual machines can be tuned with the settings in the environment profile used to deploy the pattern and the settings in the cloud group the pattern instance runs in.

The influence of these environment profile and cloud group settings on the behavior of the virtual machines is seen in two respects:

- **Prioritization**: This is the importance of each virtual machine, relative to that of all the other virtual machines in the same cloud group.
- **Resource requirements**: This is the amount of resources a virtual machine consumes from the pool allocated by the profile and provided by the cloud group. It depends on what the virtual machine says it requires and how the cloud group accounts for those requirements.

The following sections describe how prioritization and resource requirements influence the behavior of the virtual machines.

Prioritization

The individual virtual machines in deployed patterns include prioritization settings. These settings become relevant during times of resource contention, which is when a cloud group's virtual machines require more resources than the cloud group has available. Resource contention can occur in these situations:

- Deploying multiple pattern instances concurrently.
- Assigning runtime resources that are over allocated and overloaded.
- Moving VMs from one compute node to another.

When these situations occur, the system gives preference first to the higher priority VMs.

PureApplication System prioritizes a VM based on two settings:

- **Profile deployment priority**: This is the deployment priority specified in the environment profile used to deploy the pattern. This value is set by the administrator who configures the environment profile. Possible values are Platinum, Golden, Silver, and Bronze.
- **Deployer deployment priority**: This is the priority specified in the pattern deployment properties dialog used to deploy the pattern, including selecting the environment profile, as
shown in Figure 7. This value is set by the deployer who deploys the pattern. Possible values are High, Medium, and Low.

All VMs in the same pattern instance have the same priority because they are all deployed together.

**Figure 7. Priority setting in the pattern deployment properties dialog**

The combination of settings, in order of priority, is shown in Table 1.

**Table 1. Priority weighting of workloads**

<table>
<thead>
<tr>
<th>Priority</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platinum-High</td>
<td>16</td>
</tr>
<tr>
<td>Golden-High</td>
<td>12</td>
</tr>
<tr>
<td>Silver-High</td>
<td>8</td>
</tr>
<tr>
<td>Platinum-Med</td>
<td>8</td>
</tr>
<tr>
<td>Golden-Med</td>
<td>6</td>
</tr>
<tr>
<td>Bronze-High</td>
<td>4</td>
</tr>
<tr>
<td>Silver-Med</td>
<td>4</td>
</tr>
<tr>
<td>Platinum-Low</td>
<td>4</td>
</tr>
<tr>
<td>Golden-Low</td>
<td>3</td>
</tr>
<tr>
<td>Bronze-Med</td>
<td>2</td>
</tr>
<tr>
<td>Silver-Low</td>
<td>2</td>
</tr>
</tbody>
</table>
The system's internal processes run with a weight of 20, so they take priority over all user workloads.

This prioritization becomes especially relevant during failover. For example, if a compute node fails, the system recovers those VMs by restarting them on other compute nodes in the same cloud group. The VMs are moved and restarted in priority order, which means that the system recovers the VMs of the higher priority pattern instances faster so those VMs experience shorter downtimes than the lower priority ones. Also, if the target compute nodes do not have enough capacity for all of the failed VMs, then the lower priority VMs are not restarted. An administrator has to resolve this situation manually, typically by stopping some pattern instances to make resources available and restarting the failed ones.

Resource requirements
The individual virtual machines in deployed patterns include resource requirements settings. These specify the resources that the VM requires to run properly:

- **CPU count**: This is the number of virtual CPUs assigned to this VM.
- **Virtual memory (MB)**: This is the amount of virtual memory assigned to this VM.

The accounting for the CPU count depends on the type setting of the cloud group the VM runs in. For example, if the VM requires four virtual CPUs:

- A dedicated cloud group assigns the VM four physical CPUs.
- An average cloud group assigns the VM one physical CPU.

The numbers for these resource requirement settings should be higher for a VM that is expected to support a significant user load. If a VM's numbers are lower and it experiences high user load (and assuming the pattern instance does not load balance to other VMs), then this VM's users will experience degraded performance because it does not have enough resources to serve all of the requests with the customary response times.

Why not simply assign an overabundance of resources to all of your pattern's VMs? Because that curtails the number of pattern instances you can deploy and run successfully. Think of a VM as taking up space. The bigger each VM is, the fewer of them that fit. When you deploy a pattern, its resource totals are subtracted from the environment limits set by the environment profile. Once the limits for a profile reach zero, you cannot use that profile to deploy any more patterns until some of its pattern instances are stored or deleted. If you deploy patterns via separate environment profiles and over allocate the resources in the cloud group, VMs with overly generous resource settings make the over allocation even greater.

When the cloud group's resources are over allocated and the VMs all try to use their allocations, the potential problem turns into an actual problem as over allocation becomes resource contention. The cloud group's resource contention management resolves the contention using prioritization and rationing. The lower priority instances either do not start, or if they are already running, receive less than all of the resources they require.
Therefore, when setting the resource requirements for the VMs in a pattern, you need to find the sweet spot between two opposing constraints:

- **Optimize application performance**: Assign the pattern's VMs at least enough resources so that it runs adequately.
- **Optimize resource utilization**: Assign the patterns' VMs at most the resources they need to run properly under expected load. This maximizes the number of pattern instances that you can deploy with an environment profile and can run in a cloud group.

Too few resources and your application's performance will suffer. Too many resources and you cannot deploy as many applications and resource utilization will be lower.

**PureApplication System environment strategy**

Let's consider some approaches on how to use these PureApplication System resource types to fulfill these cloud computing concepts in some common scenarios.

For all of these scenarios, remember the purpose of the two main PureApplication System resource types:

- Cloud groups represent different physical deployment environments that pattern instances can run in. Cloud groups are physically isolated: You can stop a cloud group or it can fail without affecting the others. If a workload in a cloud group were to somehow go crazy and consume all of the resources, that affects the other workloads in that cloud group but it does not affect the workloads in the other cloud groups.
- Environment profiles represent different logical deployment environments. They are targets for deployment that define policies for a set of deployers. Two user groups that have the same policies and share the same resource allocations should share an environment profile (that is, both are assigned to it). Two user groups which should have different policies or get separate resource allocations each need their own environment profile.

**Scenario: Development lifecycle environments**

One common approach for defining environments is to separate stages in the application development lifecycle. Lifecycle stages, and their corresponding runtime environments, typically include:

- **DEV**: This is used for developing business applications.
- **TEST**: This is used for testing applications.
- **PROD**: This is used for running applications for use by business or end users.

Each environment typically runs on independent sets of hardware. Part of the motivation is to prevent problems that occur in the development environment from affecting the test environment and test problems from affecting production.

In PureApplication System, to create these three runtime environments and isolate them from each other, a good practice is to create three cloud groups: Dev, Test, and Prod. On a small configuration Intel system (which is one with six compute nodes), the configuration might be:
• **A Dev cloud group with one compute node** (which, for an Intel compute node, gives developers 16 physical cores): Set its type to "average" since development applications are frequently unused and receive low user load.

• **A Test cloud group with two compute nodes**: They should reside in two different chassis and different sides of the rack. Set its type to "dedicated" to mimic production.

• **A Prod cloud group with three compute nodes**: They should be distributed across the three chassis using both sides. Set its type to "dedicated" since production applications are expected to be used heavily.

How hardware is arranged in the PureApplication System rack, such as compute nodes being housed in chassis and Intel compute nodes being stacked in two columns that are powered separately, is a detailed topic. For an overview of the system's hardware details, see [A tour of the hardware in IBM PureApplication System](https://www.ibm.com/developerWorks/developerworks/library/pureappsystem/).

Each of these three cloud groups also needs at least one IP group with an otherwise unused VLAN ID to keep their network traffic separated.

As for how many environment profiles to create, there are no hardware limitations so the sky's the limit, though some practical guidelines indicate what is needed. Typically, the set of users who can deploy patterns for PROD is smaller than the set of deployers for TEST, which is smaller than the number for DEV, as shown in Figure 8.

**Figure 8. Relative number of deployers per environment**

![Figure 8](https://www.ibm.com/developerWorks/developerworks/library/pureappsystem/images/fig8.png)

Likewise, the number of profiles needed for each runtime environment tends to decrease from DEV through PROD.

First, it is helpful to define a user group for workload "super users":

• **Workload Administrators**: This is a group of workload super users who administrate the full set of workloads within the system (typically users with the Workload resources administration security role) and should be able to deploy patterns using any profile. Assign this group to every environment profile.

Here are some helpful profiles. Each profile has a corresponding user group.
• **Production application environment profiles:** These deploy applications into the PROD environment. Set their priority to golden and optionally set the environment setting to "Production".
  - You can create separate profiles for each production application, or per department or line-of-business deploying production applications, which enables the settings to control who can deploy the patterns for each application, and to allocate resources differently for different applications or groups of applications.
  - Then again, it may be better to have one set of users responsible for deploying all applications into production, in which case they all use the same profile. One consequence of deploying all applications with the same profile is that all of the pattern instances share the profile's allocation of resources. If different sets of applications should draw from different allocations of resources, create separate profiles and assign the single production deployment user group to all of those profiles.

• **Test application environment profiles:** These deploy applications into the TEST environment. Set their priority to silver and optionally set the environment setting to "Test". Create one profile per team deploying one or more applications to be tested. Using a separate profile for each team and assigning each profile separate resources, such as a separate IP group, helps keep that team's applications isolated within the cloud group, and prevents one team from using up too much of the cloud group's resources so that not enough remains available for the other teams.

• **Development application environment profiles:** These deploy applications into the DEV environment. Set their priority to bronze and optionally set the environment setting to "Development". All developers can share one profile, but then each has to be trusted not to use too many resources. To enforce these limits, use a separate profile for each development team, or even each developer. Keep in mind that a profile is only useful if it has settings that are different from other profiles, such as different settings for who can deploy patterns, what IP groups to use, or to enforce limits on resources like CPU and PVUs.

**Scenario: Multiple production environments**

Another common approach might be to isolate multiple production environments. They are all used for production applications, so they all have equal priority. However, their applications are used for different purposes and so should be isolated from each other.

For example, consider these three hypothetical production environments:

• **Public web site:** This hosts the web applications that customers access via the Internet. It is accessible by an enterprise's customers.
• **Internal HR applications:** This hosts the applications used by the HR department. It is accessible by the department employees only.
• **Internal Finance applications:** This hosts the applications used by the Finance department. It is accessible by the department employees only.

In PureApplication System, to create these three production environments and isolate them from each other, a good practice is to create three cloud groups: PUBLIC, HR, and FINANCE. On a small configuration with six compute nodes, you can assign each cloud group two compute nodes,
spreading each pair across chassis and sides, and each with a different unused VLAN ID to use internally. Give each cloud group a one or more IP groups. The IP groups for a cloud group can have the same VLAN IDs, but the IP groups for different cloud groups need different VLAN IDs to isolate the cloud groups' network traffic from each other. Assuming the applications are used heavily, set each cloud group's type to "dedicated". Otherwise, if a cloud group has a surplus of applications that are used sparingly, set its type to "average".

Create an environment profile for each production environment and assign it a user group whose users are responsible for deploying to that environment. Also create a Workload Administrators group of super-users (administrators of all workloads) and assign it to all of the profiles. If there are different teams that share an environment, but are having trouble playing together nicely, create a different user group and profile for each team. The profiles deploy their patterns to the same cloud group, but they allocate resources with limits to help isolate the teams' applications better.

**Scenario: Utilization-based environments**

One feature of cloud groups is the type setting:

- **Dedicated**: This provides one physical CPU for every virtual CPU requested by a virtual machine.
- **Average**: This provides one-quarter of a physical CPU for every virtual CPU requested by a virtual machine.

You can use this—in conjunction with the environment limits and priorities on an environment profile and the resource requirements on a virtual machine—to create a cloud group optimized for one of two different types of workloads:

- A cloud group that runs fewer applications, but is better prepared for more of them to have a higher simultaneous user load.
- A cloud group that over allocates its CPU to run four times as many applications, enabling underutilized applications to run with higher density for improved average resource utilization.

For an organization with both high-load applications and under utilized applications, a good practice is to create a cloud group for each type.

To apply this practice, create two cloud groups:

- **High Load**: This is for applications that usually get significant user load:
  - Set the cloud group's type to "dedicated".
  - Set the environment profiles' environment limits conservatively to help prevent each user group from deploying too many patterns (which uses more than the team's fair share of limited resources).
  - If some applications have higher priority than others, set those in the environment profile and when deploying the patterns. This comes into play when the user load is greater than the cloud group's capacity.
  - Set the resource requirements in the patterns' VMs high, but only as high as they need at peak load.
• Even if the load peaks on many of these applications simultaneously, this cloud group is better prepared to handle the load and keep the response times consistent.

• **Underutilized**: This is for applications that need to be available and sometimes get a fair amount of user load, but generally are not used very heavily:
  • Set the cloud group's type to "average".
  • Set the environment profiles' environment limits liberally so that each team can deploy lots of patterns. The expectation is that they are not used much.
  • If some applications have higher priority than others, set those in the environment profile and when deploying the patterns. This comes into play when the user load is greater than the cloud group's capacity.
  • Set the resource requirements in the patterns' VMs conservatively, but high enough that they can run acceptably under a typical load.
  • This cloud group can accept a higher number of pattern deployments, with the caveat that performance suffers if the load peaks on many of these applications simultaneously.

If a user group is responsible for both high-load and under utilized applications, you can assign it to environment profiles for both cloud groups. Do **not** use a single environment profile for both cloud groups because you want separate resource limits for the two cloud groups, and you want the limits to be conservative for the high-load group, but liberal enough for the under utilized group.

With this approach, you give the high-load applications the best chance of receiving the resources they need while also getting the best resource utilization possible running applications that are not used much.

**Scenario: Shared development and test environment**

You can consider combining some of the development lifecycle runtime environments. Combining TEST and PROD is a bad idea because testing can take resources away from PROD and problems in TEST can affect PROD. It is better to isolate those environments in different cloud groups.

On the other hand, combining DEV and TEST in a single environment may make more sense. There are pros and cons to this approach.

• **Pro**: The resources not being used for one purpose can easily and dynamically be used for another. For example, when the testing effort is low, you can use those resources for development, and then shift them back to testing when that increases.

• **Con**: Heavy testing can consume most resources, limiting those available to development (assuming development is set as a lower priority). This is a pro when development ebbs during testing, perhaps because the developers are doing the testing. However, it is a con for developers trying to do development while the testing effort is significant.

To implement this approach, create one DEV-TEST cloud group. Create DEV and TEST pairs of the other resources: IP groups, environment profiles, and user groups. Of course, assign a Workload Administrators user group to both or all of the environment profiles. With these resources, make these settings in the two profiles, as shown in Table 2.
Table 2. Environment profile settings

<table>
<thead>
<tr>
<th>Setting</th>
<th>TEST profile</th>
<th>DEV profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access granted to</td>
<td>Test user group</td>
<td>Dev user group</td>
</tr>
<tr>
<td>Deploy to cloud groups</td>
<td>Dev-Test cloud group</td>
<td>Dev-Test cloud group</td>
</tr>
<tr>
<td>IP addresses</td>
<td>Test IP group</td>
<td>Dev IP group</td>
</tr>
<tr>
<td>Environment limits</td>
<td>10 virtual CPUs</td>
<td>5 virtual CPUs</td>
</tr>
<tr>
<td>Deployment priority</td>
<td>Silver</td>
<td>Bronze</td>
</tr>
<tr>
<td>Environment</td>
<td>Test</td>
<td>Development</td>
</tr>
</tbody>
</table>

The advantages of this approach are:

- **Access granted to**: Different user groups control who can use each profile to deploy the patterns.
- **Deploy to cloud groups**: Both profiles deploy to the same shared cloud group.
- **IP addresses**: Different IP groups make sure that one user group’s pattern instances cannot use up all of the addresses and leave the other user group with none. It also enables separate VLANs if desired.
- **Environment limits**: This makes sure that DEV gets the computational power of, at most, 5 virtual CPUs and TEST gets, at most, 10 virtual CPUs. If they grow to use all available CPUs, when the higher-priority environment (TEST) needs more CPU, that capacity is taken from the lower-priority environment (DEV).
- **Deployment priority**: Sets TEST with a higher priority than DEV so that resource contention is resolved in the favor of the applications deployed with the TEST profile.
- **Environment**: This is ignored by the system.

**PureApplication System environment principles**

These scenarios make consistent use of principles that tend to apply to all scenarios:

1. Use cloud groups to divide a PureApplication System into isolated logical computers. In order to isolate each cloud group’s network as well, each cloud group’s IP groups needs to have a different application VLAN ID.
2. Tune each cloud group for either high-load applications with low density or lightly used applications with high density. If you have both types of applications, create one of each type of cloud group.
3. An environment profile is typically used to deploy to a single cloud group.
4. Use multiple environment profiles to separate user groups sharing a cloud group, such as multiple teams deploying to the same testing environment. This enables the PureApplication System administrator to confine each group to use its own IP groups, priorities, and resource limits, which enables (and perhaps forces) the users and their applications to cooperate more easily.
5. Create a Workload Administrators user group for super users (those who administrate the PureApplication System) and assign that to all environment profiles.

Follow these principles and you are well on your way to using the features of PureApplication System successfully.
Conclusion

This article explained how to design and create application runtime environments in PureApplication System using its features for cloud groups and environment profiles. It showed how these features relate to cloud computing concepts, described these and related features in PureApplication System, considered scenarios for using these features, and reviewed principles that can be drawn from these scenarios. With this information, you are now prepared to administer the runtime environments in your PureApplication System.

Acknowledgements

The author would like to thank the following IBMers for their help with this article: Vishy Gadepalli, Stanley Tzue-Ing Shieh, Michael Fraenkel, Shaun Murakami, Jason Anderson, Ajay Apte, Kyle Brown, Rohith Ashok, and Hendrik van Run.
Resources

- Product information:
  - IBM PureApplication System
  - IBM PureApplication System Version 1.0 Information Center
  - IBM PureSystems Centre
  - IBM PureSystems resources on developerWorks

- Articles:
  - A tour of the hardware in IBM PureApplication System
  - Best practices using infrastructure as a service in IBM PureApplication System
  - Navigating the IBM cloud, Part 1: A primer on cloud technologies
  - Demonstrating workload isolation in IBM PureApplication System
  - Managing administrative access in IBM PureApplication System
About the author

Bobby Woolf

Bobby Woolf is a Consultant for IBM Software Services for WebSphere (ISSW), focusing on IBM PureApplication System, service-oriented architecture, event-driven architecture, and application integration. He is the author of Exploring IBM SOA Technology and Practice, and a co-author of Enterprise Integration Patterns and The Design Patterns Smalltalk Companion.

© Copyright IBM Corporation 2012, 2013
Trademarks
(www.ibm.com/developerworks/ibm/trademarks/)