DGP systems have been designed to run as simple federated computing infrastructures (two-tier client server systems using load balanced web server farms supported by data storage clusters). This basic type of infrastructure has been in use for decades, and is very stable and reliable. DGP then optionally extends the federated computing system to replace the standard storage clusters with a type of distributed data storage that converts it into a distributed grid computing system.

DGP has therefore been designed to run on "bare metal" computers and VM's (IaaS). It is possible that the web server processing nodes could be run as containers, but that has not been tested. Also, containers were not designed for any type of persistent data storage, so they are a poor fit for either the standard data storage clusters or DGP's distributed grid equivalent.

DGP client applications (and integrated systems) have no dependencies on the server tier other than the API messages used to call the web service APIs. They are therefore free to use any type of platform and programming languages desired. A similar loose coupling exists between the web service APIs and the data storage tier, depending only on the ADO.NET or JDBC drivers.

Requirements

1. All tiers of a system should be able to run on a single computer

What: *The dev environments allows all tiers of the system to run on a single computer.*

Why: *The ability to run all tiers of a system on a single computer is the best and simplest configuration for software development, testing and debugging. It is also necessary for very small-scale systems.*

Testing: *Each developer workstation is a good test platform to ensure that all of the tiers of a DGP system work correctly when running as "localhost". DGP has been designed so that each tier is loosely coupled to the other tiers, and only the configuration of the endpoints of the various RPC's determines where a given tier is run.*

2. At a larger scale the Dev, Test and Prod environments should be separated from the business network

What: *The software system should have separate dev, test and production environments. An additional QA environment may be needed to run end-to-end tests in production while avoiding the inclusion of test data in the production databases.*

Why: *At a large scale, the separate environments are a necessary prerequisite for delivering high quality software systems over time. Dev and Test environments can be subnets of the business network, but QA and Prod environments should be completely separate due to the very strict security needed in the Prod environment that is not compatible with the needs of the business network. These requirements can usually be relaxed for small-scale systems.*

Testing: *Restricted access to the various subnets and separate networks can be tested by the different groups of technical staff. Architects, developers and testers should only have access to Dev and Test environments (not QA or Prod).*

3. At a larger scale, the production environment should be divided into a DMZ and an internal network that protects both the processing and data storage nodes.

What: *The environments are formed by firewall rules into separate subnets that can only be accessed sequentially.*

Why: *Access from external networks is limited to the DMZ, which contains no significant logic or data. Only the servers in the DMZ are allowed to access the web service APIs of the internal network (and nothing else). Also, logic that calls out to external networks is only allowed to run in the DMZ. This allows any attempt to call external networks from the internal network to be more easily detected and traced to its source.*

Testing: *These firewall rules can be tested using the API Test Harness and/or Postman to make calls into or out from the various networks.*

4. DGP should be able to easily migrate from cloud IaaS VM's to running on premise or in a collocation as needed.

What: *The types of architecture and tools used to build DGP should be compatible between the cloud and a collocation.*

Why: *The most cost-effective infrastructure for the business changes with the scale of operations. At smaller scales, cloud infrastructure is the lowest cost option. However, the cloud resources are the most expensive, and as systems grow, the cost of the cloud resources also grow rapidly. Eventually, the system will be able to significantly lower costs for the*

business by switching from the cloud to a collocation infrastructure. For this reason, it is a good idea to use architectures and tools that are compatible between the cloud and a collocation (or on-premise).

Testing: *IaaS is the infrastructure type that is compatible between local workstations, on premise servers and VM's, collocation servers and VM's, and cloud VM's. Using a combination of host computers and VM's proves the compatibility of the software and associated processes (CI-CD-CT, etc.) between those various options.*

5. Vendor lock-in to proprietary solutions should be avoided whenever possible.

What: *The use of proprietary tools, services and solutions that lock a software system into their use should be avoided as much as possible.*

Why: *Vendor lock-in is most common when using proprietary cloud PaaS and SaaS solutions. This type of lock-in would prevent the migration of the system from the cloud to any other type of infrastructure such as a collocation, or even prevent the migration to a different cloud vendor.*

Testing: *Using the IaaS infrastructure type will prevent vendor lock-in, and VM's hosted on different cloud providers is one way to prove that cross-infrastructure and cross-vendor compatibility works well in practice.*