

Local Databases:

From the diagram Developer is Mac and the VM is of Vagrant type: Ubuntu 16.04 and there were some code repo’s in dyson corresponding webfolders belonging to different regions so that with help of nginx tool it can pull the code-repo and the database is of type MariaDb and data also belonging to different regions similar to that of code repo’s

Note: The Database is in Vagrant VM type so if we destroy and try to bring back a VM all the data in MariaDb gets destroyed and leaving the code repo’s unaffected. so we need to perform a necessary backup for the database

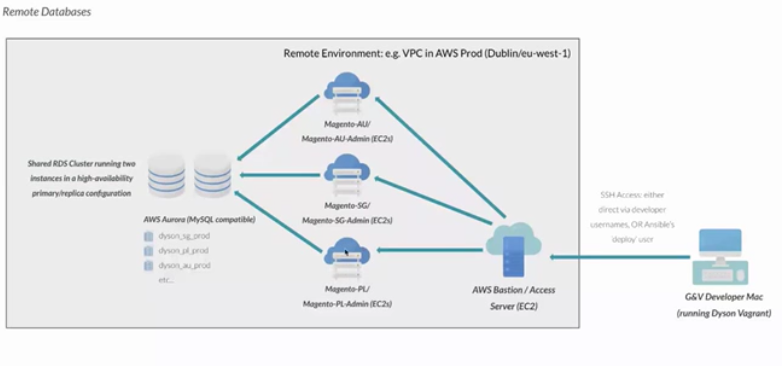
The database and nginx can communicate both on same page when u’re browsing local developer sites

Dyson CLI is the tool which interacts with Dyson remote environments and using it we can pull/push databases around safely to remote environments

The data hasn’t been sanitized in local environment and it’s been sanitized in Dyson remote environment

The other way we may access databases is by using Sequelpro, Mysql workbench etc.

Remote Databases:



Accessing Remote Databases:

There is no access to RDS or any EC2’s from outside VPC except via Bastion

To connect to remote RDS hosted database, you must SSH into the Bastion and tunnel to valid website EC2 in VPC

From there, security groups will allow access to the RDS, should you have a valid MYSQL user

Dyson Vagrant ships with SSH config means you will automatically SSH tunnel into EC2 by SSHing to it’s private IP address, SSHing EC2’s will actually SSH into Bastion and then forward you on

Ansible roles are equipped to use market’s main database user, you don’t need to know how to use it, Ansible has it’s own user called ‘deploy’ which uses the developers SSH key

If you wish to configure access via a desktop database viewer such as Sequel Pro or MYSQL workbench, you configure it to connect via SSH into that market’s EC2 and use MYSQL credentials for that market.

Interacting with databases:

Dyson CLI will use the database tools for db:push and db:pull unless it’s necessary

These tools utilizes a third-part Magento CLI called n98 Magerun2, it’ll be installed automatically and will run on remote and local environments

Common tweaks to standard tables:

For the most modules with the custom tables, there’ll be scripts of the InstallSchema, InstallData, UpgradeSchema and UpgradeData for each module

Tables were named related to their module name eg: Dyson soft products were prefixed as ‘catalog\_product\_soft\_tablename’

Most changed standard table was ‘sales\_order’ table in the module and it often gets a schema upgrade by whatever payment gateaway is installed e.g: ‘adyen\_notification\_event\_upgrade’

Setup and Upgrade Data:

In Magento we don’t edit or change databases unless a Magento upgrade/installation does this or through InstallSchema, InstallData, UpgradeSchema and UpgradeData scripts included with custom modules

Any custom modules can have it’s setup scripts checked for full history of what databases changes were made by the module

InstallData and InstallSchema are for first time installations and can create/modify databases and populate with any critical data, if you have a module installed, these two scripts will never run again

UpgradeData and UpgradeSchema do the same thing but changes can be version managed, When a module update needs new schema or data changes, this is done in these two scripts, if you upgrade the module, these two scripts will perform and run action based on version you are upgrading, This is all standard Magento functionality.

Commerce vs Open Source Databases:

Previously commerce splitted databases and it’s deprecated now

Commerce builds are safe from needing to migrate to a single database going forward which will be the case at some point

Magento commerce adds many of it’s own additional tables as a result of extended functionality offered in Commerce core

Whereas Open source database can be upgraded to upgrade with Commerce but here we’re using Commerce databases.