Local Databases: Here developer is using Mac and the VM is of Vagrant type: Ubuntu 16.04 and there were some code repo’s in dyson corresponding web folders belonging to different regions so that with help of Ngnix tool it can pull the code-repo and the database is of type Maria DB and data also belonging to different regions similar to that of code repo’s.

The Database is in Vagrant VM type so if we destroy and try to bring back a VM all the data in Maria DB gets destroyed and leaving the code repo s unaffected so we need to perform a necessary backup for the database. The database and Ngnix can communicate both on same page when we are 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.

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 validate website EC2 in VPC. From there, security groups will allow access to the RDS, you should have a valid MYSQL user. Dyson Vagrant ships with SSH config means you will automatically SSH tunnel into EC2 by doing SSH to its private IP address, by doing SSH 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 its 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: For the most modules with the custom tables, there’ll be scripts of the Install Schema, Install Data, Upgrade Schema and Upgrade Data for each module. Tables were named related to their module name e.g.: 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 gateway 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 Install Schema, Install Data, Upgrade Schema and Upgrade Data scripts included with custom modules. Any custom modules can have its setup scripts checked for full history of what databases changes were made by the module. Install Data and Install Schema 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. Upgrade Data and Upgrade Schema 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 were 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 its own additional tables as a result of extended functionality offered in Commerce core. Whereas Open source database can be upgraded but here we’re using Commerce databases.