Database Platform as code@PayPal

05/21/2020

Agenda

1. Introduction
2. PayPal's Need for Automted Framework
3. PayPal's Framework
4. Insight into DBCOPY flow
5. PayPal’s DB Build requirements
6. PayPal’s DB Build workflow
7. What’s Next

About me

- Database Engineer at PayPal for 10+ Years
- Working on ORACLE/Unix Technologies for 19+ years
- Interested in Automation and handy with shell/perl/python
- www.linkedin.com/in/indhar

About PayPal

Two decades ago, our founders invented payment technology to make buying and selling faster, secure, and easier; and put economic power where it belongs: **In the hands of people**

Our 300+ Million consumers can accept payments in > 100 currencies and interact with 20M+ Merchants across 19K+ corridors

Almost 8000 PayPal team members provide support to our customers in over 20 languages

We are a trusted part of people's financial lives and a partner to merchants in 200+ markets around the world

Database Infrastructure & Storage Footprint

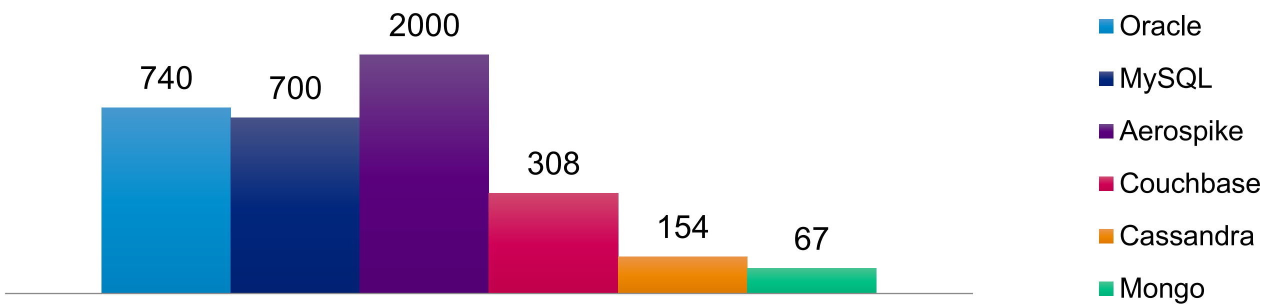
5M+
Execs/Sec

750+
ORACLE Instances

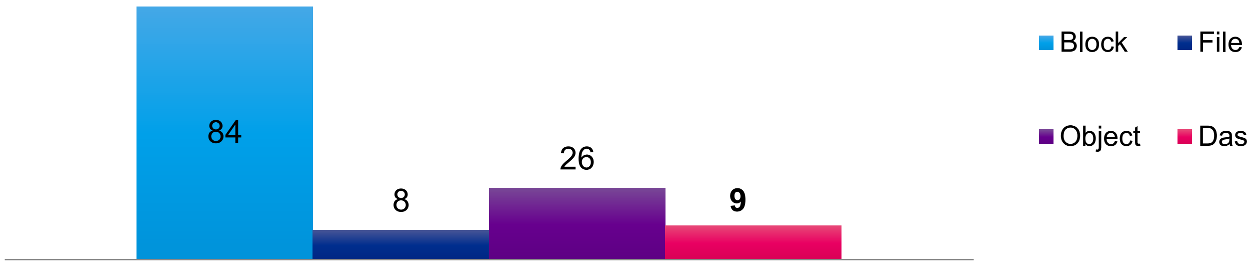
32% Y-o-Y
DB Storage Growth

93 PB
Total DB Storage

Host Count by Database Type



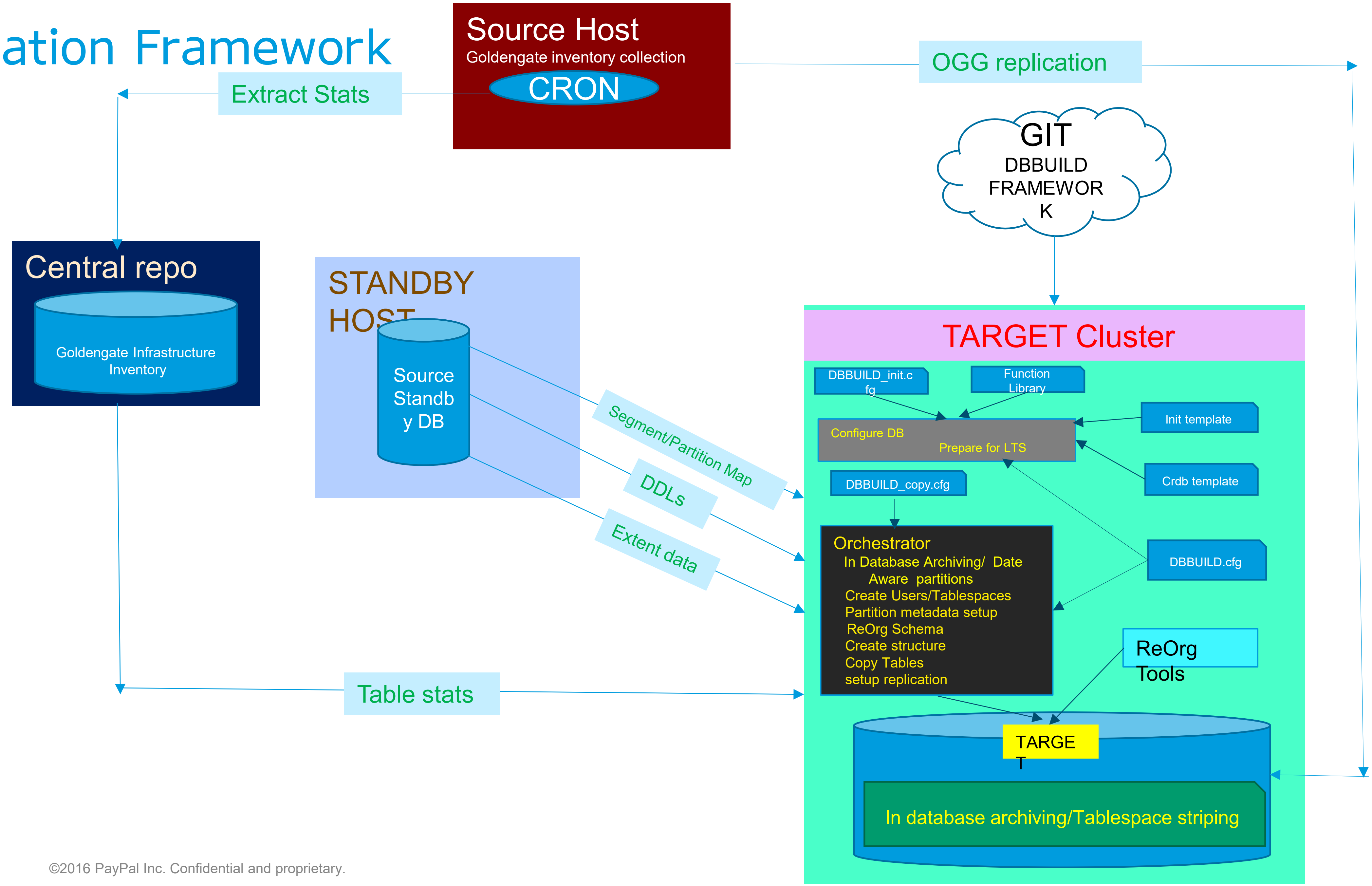
Storage Footprint (PB) by Type (Utilization)



Need for Automated Framework

Size and Scale	Sheer size of the Databases being built and reduce repetitive tasks
Minimal Input	Human intervention to be reduced drastically
Standardization	Reduce scope for subjective/accidental configuration errors between different instances of Build
Time to Deliver	Reduce Time To deliver
Complexity	Abstraction of complexity of build
Version controlled	Version controlled deployment
In-database archiving	Implement in-database archiving to support storage tiering/ offloading

Automation Framework



Source Host

Goldengate inventory collection

CRON

Extract Stats

OGG replication

GIT

DBBUILD FRAMEWOR K

Central repo

Goldengate Infrastructure Inventory

STANDBY HOST

Source Standb y DB

Segment/Partition Map

DDLs

Extent data

Table stats

TARGET Cluster

DBBUILD_init.c fg

Function Library

Init template

Configure DB

Prepare for LTS

Crdb template

DBBUILD_copy.cfg

DBBUILD.cfg

Orchestrator

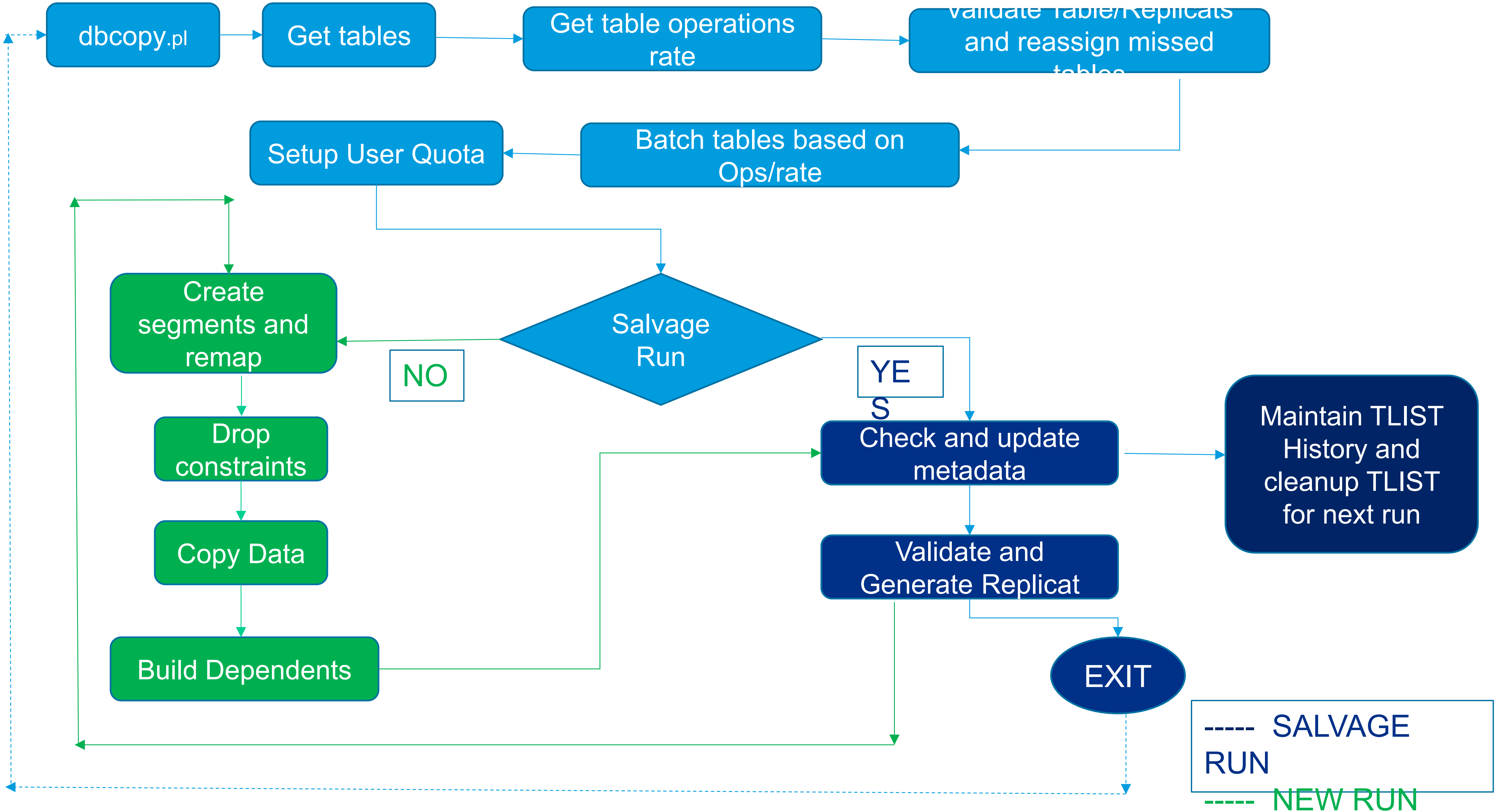
- In Database Archiving/ Date Aware partitions
- Create Users/Tablespaces
- Partition metadata setup
- ReOrg Schema
- Create structure
- Copy Tables
- setup replication

ReOrg Tools

TARGE T

In database archiving/Tablespace striping

Insight into DBCOPY flow


dbcopy.pl
Get tables
Get table operations rate
Validate Table/Replicats and reassign missed tables
Batch tables based on Ops/rate
Setup User Quota
Salvage Run
NO
YES
Create segments and remap
Drop constraints
Copy Data
Build Dependents
Check and update metadata
Maintain TLIST History and cleanup TLIST for next run
Validate and Generate Replicat
EXIT
----- SALVAGE RUN
----- NEW RUN

DB Build Requirements

- Hardware configuration
  - Memory, Cores
  - Media Drive, Capacity
  - Compute-Storage mapping
  - Pooling Storage, Compute

- Software configuration
  - Memory, Sessions profile
  - Objects
  - Tablespace layout
  - Monitoring
  - Out-of-the-box archiving

DBBUILD_copy.cfg_template

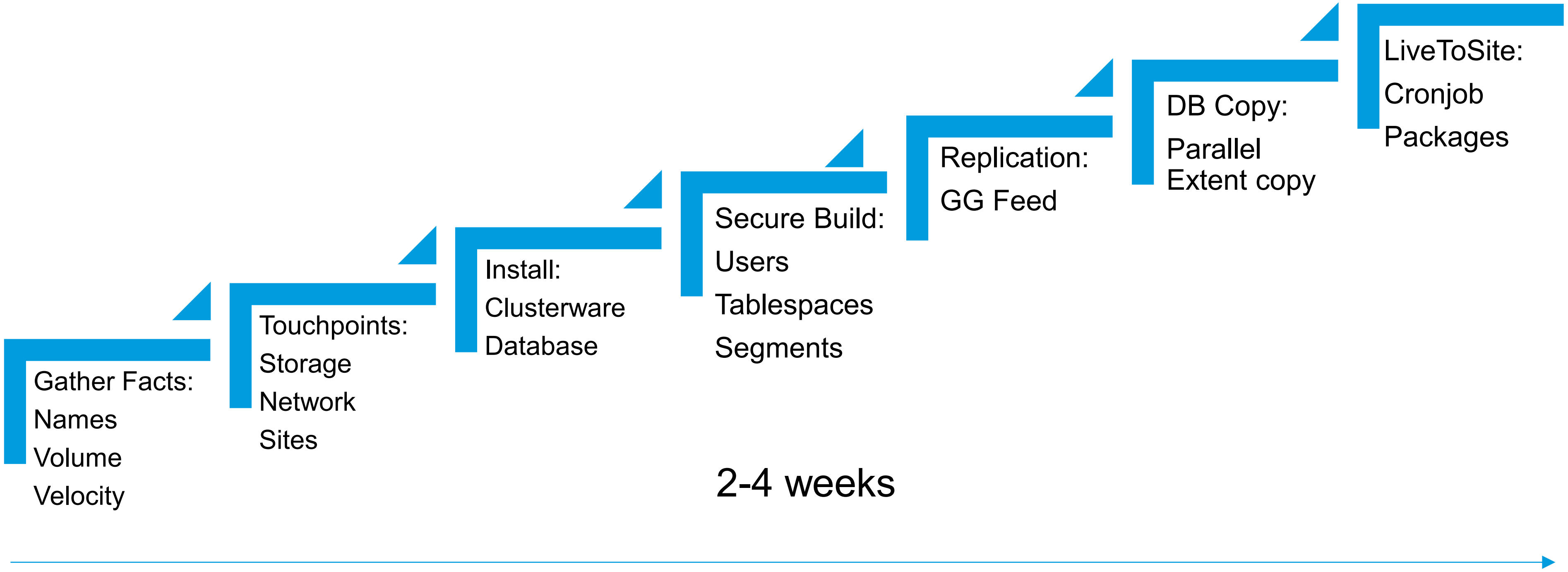
```
 DBNAME=                    ## The DB family
 HOSTING_MEMBERS=           ## Comma seperated Hosting members, generally without FQDN
 DB_UNIQUE_NAME=            ## The unique name to be assigned to this DB
 PDB=                       ## Current PDB container name
 SRC_USERS=                 ## SRC_USERS can either be ALL ( excluding ORACLE default users ) or a comma separated
list
 COMPRESSION=               ## Compression type
 REMAP_SCHEMA=              ## REMAP_SCHEMA should be comma separated pairs of SRC:DEST schema in case we need
remapping
 EXTRACTS=                  ## the Source extracts separated by ",”
 GGS_HOME=                  ### GG Home
 MAX_REPLICAT_RATE=                  ## Maximum permissible txn rate for Replicat
 MAX_TBSPC_USED_SIZE_TB=                 ## Max Tablespace Occupancy ( in TB) - used for striping across tablesapces
 SRC_DBNAME=                ## Source DB Family name
 SRC_DB_UNIQUE_NAME=                 ## Source DB PRIMARY Unique name
 SRC_STANDBY=                        ## Source Standby that we would use for DATA COPY purposes
 STBY_THREADS=                       ## # of DBCOPY threads to spawn
 PQ_QUERY_THREADS=                   ## # of parallel query threads - MOSTLY used in IOT
 TGT_BUILD_THR=                      ## # of parallel BUILD threads
 PQ_BUILD_THREADS=8              ## # threads to use in insert /*+ APPEND parallel */ while building HEAP/IOT tables
 INDEX_THREADS=16                ## # of index segments to rebuild parallelly
 INDEX_DDL_DEGREE=4              ## # PQ degree to be used for each of the above threads
 BODHI=<TNS for the DB>          ## BODHI DB TNS

#OVERRIDE Parameters
 OVERRIDE_DATADG=
 OVERRIDE_FRADG=
```

[Contd.] Template Files Visualization

Format : PARAMETER PATTERN = <tagged value>

```
processes=<PROCS>
cluster_interconnects=<CLUST_INTCON_SID>
control_files=<CTRL_FILES>
undo_tablespace=<UNDOTBS_SID>
instance_number=<INST_NUM_SID>
instance_name=<INST_SID>
sshared_pool_size=<SPSZ>
streams_pool_size=<STRMPLSZ>
pga_aggregate_target=<PGASZ>
cluster_database=<CLUDB>
cluster_database_instances=<CLUINST>
db_file_name_convert=<N>
fal_client=<N>
fal_server=<N>
db_create_file_dest=<N>
```

DB Build Workflow



Gather Facts:
Names
Volume
Velocity

Touchpoints:
Storage
Network
Sites

Install:
Clusterware
Database

Secure Build:
Users
Tablespaces
Segments

Replication:
GG Feed

DB Copy:
Parallel
Extent copy

LiveToSite:
Cronjob
Packages

2-4 weeks

What’s next

- ❖Fully Automated Standby Builds
- ❖Remote Deployments through Ansible
- ❖UI based Build-outs - DSaaS
- ❖Performance/Capacity monitoring framework
- ❖Automated instance placements – Exadata server Farm

Q&A