Answer 3:

Installations:

1. **Crunchy Postgres operator:** Using the OpenShift OperatorHub install the Crunchy Postgres operator version 5.4.3.

2. **Postgres instance:** Install Postgress version 15 using the Postgres operator installed above.

3. **Use Flyway to extract test schema:** Use Flyway to ensure the test data matches the existing production database.

Data Engineering:

1. **Data Engineering:** Since this is a test database, we can use a specific percentage of data instead of using the entire data as that would require production grade nodes and compute. Let us suppose we take all 4.5 million rows to evaluate. Since, the data mostly consists of text, hashes, or number id's we can assume certain data sizes per row across different tables:

housing	500 bytes/row
housing_type	50 bytes/row
ownership	500 bytes/row
owners	500 bytes/row

This translates to a total of 2.25 GB. This amount of data can be easily stored in any node with just 8 GB RAM and 2-4 vCPUs. We therefore do not need to sample data as this small node can also handle 3x the number of rows and keep all rows in memory. If required OpenStack can be easily used to scale these databases vertically (easier for SQL databases) and horizontally as well using the Horizontal Pod Autoscaler (HPA).

2. **Data Usage:** To use the data we need to perform 2 steps – extracting the data from the production setting and importing it to the test namespace. We use *pg_dump* and *psql* to achieve both these operations respectively:

```
pg_dump -h {production-db-hostname} -U {username} -d {production-db-name} -t housing -t housing_type -t ownership -t owners --data-only > test_data.sql
```

All databases are separated by -t while other information is self-explanatory.
test_data.sql stores the entire database or a subset (not required so far) as a with a *.sql* extension.
This file can then be imported to the test environment using:

```
psql -h {test-db-hostname} -U {username} -d {test-db-name} -f test_data.sql
```

3. **Data Configuration:** All environment variables should be pointing to the new test-postgress instead of the production-postgress. This requires *yml* changes for decoupling application code from operational overhead. For example, *docker-compose-prod.yml* and *docker-compose-test.yml* with separate instances should be configured.

4. **Data Management:** Define a Persistent Volume Claim (PVC) and spin up a Persistent Volume (PV) for the new test postgres database. Since the data is still small 2x space (~6 GB) as compared to the memory should be a good starter.

Test Strategy:

1. **Integration test:** Performing integration tests will ensure that the switch works properly across environments.

2. **Data refresh:** Data needs to be refreshed to ensure that the test environment remains representative of the production environment. The above commands and installations can be automated to make sure end to end workflows work with the newest data.

3. **Further testing:** The test environment also provides exclusive rights to perform stress tests as well as test data breaches or leaks since some of the data includes phone numbers etc. which could be critical and might require anonymising in the test environment.