**Study Guide: Great Expectations CLI & Enterprise Features**

This study document provides a comprehensive understanding of the CLI tools and enterprise-level capabilities offered by **Great Expectations**—a powerful open-source data validation framework.

**1. Overview of Great Expectations**

Great Expectations (GE) helps validate, document, and profile data as it flows through pipelines. It provides rich support for data quality checks, seamless integration with workflow orchestration tools, and visibility into data assets.

**Key Capabilities:**

* Create & manage Expectation Suites
* Run data validations through Checkpoints
* Auto-generate documentation
* Integrate with orchestration and observability tools

Supported Environments:

* Pandas, PySpark, SQLAlchemy
* Cloud Storage (AWS S3, Azure, GCS)
* CI/CD tools & Notebooks

**2. Great Expectations CLI**

The CLI is the core interface to create and manage GE projects and assets.

**Essential CLI Commands:**

* great\_expectations init – Initializes a GE project structure.
* great\_expectations suite new – Create a new Expectation Suite interactively.
* great\_expectations suite scaffold – Generate expectations based on data profiling.
* great\_expectations checkpoint new – Create validation checkpoints.
* great\_expectations docs build – Build or rebuild data documentation.

**Benefits:**

* Streamlined interaction with GE configuration.
* Version control and reproducibility.
* Easy automation via scripts or job schedulers.

**3. Auto-Profile Inference**

GE includes tools for automatic inference of expectations using sample data.

**Process:**

* Use suite scaffold to analyze a batch of data.
* GE infers expectations like:
  + Column null ratios
  + Value ranges (min, max)
  + Uniqueness and pattern matches
  + Datatype checks

**Advantages:**

* Quick setup for new data sources.
* Establishes a baseline for manual refinement.
* Ideal for onboarding unfamiliar datasets.

**4. Expectation Rules: Regex, Range, Completeness**

**a. Regex Rules**

expect\_column\_values\_to\_match\_regex(column="email", regex="^[a-z0-9.\_%+-]+@[a-z0-9.-]+\\.[a-z]{2,}$")

* Validates formatting of text fields like emails or identifiers.

**b. Range Rules**

expect\_column\_values\_to\_be\_between("sales", min\_value=0, max\_value=10000)

* Ensures numeric columns stay within expected bounds.

**c. Completeness Rules**

expect\_column\_values\_to\_not\_be\_null("customer\_id")

* Guarantees essential fields are always filled.

**Use Cases:**

* Data cleansing
* Schema enforcement
* Fraud detection

**5. Checkpoint Scheduling**

Checkpoints define when and how validations are executed.

**Definition Formats:**

* YAML files (declarative)
* Python dictionaries (programmatic)

**Scheduling Methods:**

* Airflow DAGs
* Azure Data Factory pipeline steps
* Cron jobs
* GitHub Actions / CI tools

**Command:**

great\_expectations checkpoint run my\_checkpoint\_name

**Benefits:**

* Ensures validation is integrated into ETL workflows.
* Helps in maintaining data SLAs.
* Supports multiple suites and batch runs.

**6. Webhook Alerts**

Validation results can trigger alerts to notify teams in real-time.

**Supported Channels:**

* Slack
* Microsoft Teams
* PagerDuty
* Custom webhooks (HTTP POST)

**Checkpoint Action List Configuration:**

- name: send\_slack\_notification\_on\_validation\_result

action: great\_expectations.action.SendSlackNotificationAction

**Use Case:**

* Alert DataOps teams of validation failures.
* Trigger downstream recovery processes.

**7. Source→Target Mapping**

In modern data architectures, data flows through multiple layers:

**Example Mapping:**

* **Bronze Layer** → Raw data
* **Silver Layer** → Cleaned/transformed data
* **Gold Layer** → Curated/reporting data

**Validation Strategy:**

* Run expectations at each layer.
* Validate transformations between source and target.
* Use tags or naming conventions for clarity.

**Purpose:**

* Detect transformation errors.
* Compare row counts, business rules pre/post transformation.
* Enforce data consistency.

**8. Azure Purview Lineage Registration**

**Purview** is Microsoft’s data governance and cataloging service. Integration with GE enhances observability and governance.

**Benefits of Registration:**

* View datasets and their validation results in Purview UI.
* Understand lineage and data quality together.
* Enhance compliance, audit readiness, and impact analysis.

**Integration Methods:**

* Python script to register datasets & expectation suites.
* Use Azure Functions or Logic Apps to trigger registration.
* Push metadata via REST API post-validation.

**Metadata Fields Registered:**

* Dataset name and location
* Expectation suite names
* Checkpoint run history and results
* Validation timestamps and outcomes

**9. Summary & Best Practices**

* Use CLI for fast onboarding and scripting.
* Auto-profile for quick insights into new data.
* Apply regex, range, and completeness rules to enforce business quality.
* Use scheduled checkpoints to align with ETL jobs.
* Configure webhooks to ensure proactive monitoring.
* Map validations across data pipeline layers.
* Register metadata in Purview for lineage and governance.

**10. Further Reading and Tools**

* [Great Expectations Documentation](https://docs.greatexpectations.io/)
* [Azure Purview](https://learn.microsoft.com/en-us/azure/purview/)
* [Slack Alert Actions for GE](https://docs.greatexpectations.io/docs/terms/action/)
* GitHub Repositories for examples and connectors