Progress® Chef®

Courier Product Guide

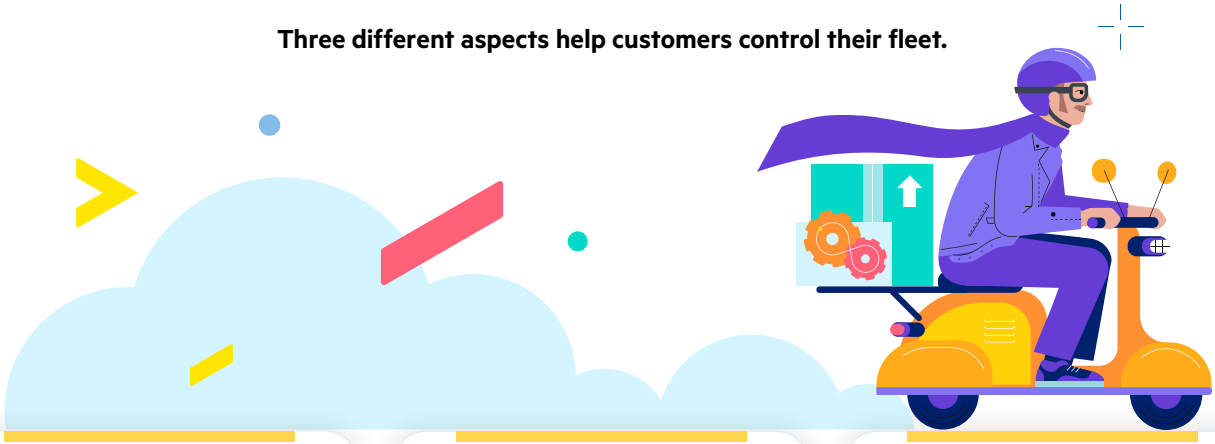
PRODUCT GUIDE

IT operators, Infosec and DevOps engineers often struggle with unplanned and reactive work. Most of this is due to a lack of on-demand orchestration, leading to ad hoc requests being serviced one-off. This can lead to patching, app deployment, incident response and an overall reduction in productivity and confidence.

Progress® Chef® Courier™ is a job orchestration platform that lets you address everyday impromptu distractions. It is a set of Chef services that allow users to orchestrate timing and targeting dimensions for the action/execution capabilities that all other Chef products provide. Courier is a job orchestration and automation tool for executing and verifying your actions.

Courier gives you complete control and flexibility over how customers can perform actions on the fleet, enabling IT operators, InfoSec engineers and other administrators to run on-demand jobs. Adding the ad hoc job orchestration feature to the existing infrastructure management offering eliminates the need to build custom solutions for emergency tasks and reactionary work.

Three different aspects help customers control their fleet.



Whenever

When you want to perform an action – now, on-demand, scheduled, recurring or recurring at a specific interval with exception.

Wherever

When you want to target a specific node or a subset of nodes, e.g., start with staging, production or a particular region, such as the US east coast.

Whatever

When you want to perform actions, such as an OS command, a cookbook run, an InSpec scan or a combination. The output of one command can be passed to another command, or it can be conditional, where a command runs if the previous run succeeds.

Terminologies

Dispatcher	Runner	Skill	Node	Interpreter
Server-side configuration UI, repository and service that provides JSON definitions to request Runners on demand.	Client/ agent microservice that requests job definitions from Dispatcher and uses those definitions to execute actions initiated from the memory state and CPU threading it is operating under (or "installed on" in the physical server world).	Skill is the ability to perform actions for a particular outcome that may or may not be installed and managed by Chef (example: Chef Infra, Courier runner).	An individual component of your system, physical or virtual (e.g., server, workstation, IP router, virtual machine or another node or component) that is assessed, installed, configured, updated, scanned and/or managed by any skill.	The runner invokes individual services to execute each type of step in the actions. Every skill will have a corresponding interpreter to work with the Courier.

Fundamentals of Job

• **Courier job**: The single place in which What, Where and When are provided for actions to be performed. This is an umbrella term that is to be used at the highest level for Courier. Users can define multiple jobs. A job can be executed numerous times across various nodes for multiple actions.

• **Job definition template:** This is the fundamental specification of a courier job, a JSON text containing the details of a single courier job.

• **Job instance:** A single occurrence of a job. There can be multiple instances for every job. An instance can be executed for various actions on multiple nodes.

• **Job run:** The assigned job instance for a node. Every node provided in the job definition will have a job run for every job instance, which is across multiple nodes.

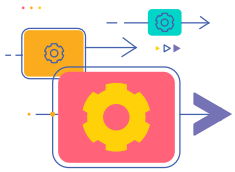
• **Action:** Every node is to execute the workloads with corresponding payloads during a job run.

• **Step:** Individual commands to be executed for an action of a job run. OS command action can have multiple steps, such as making and reading files.

How you can use Courier

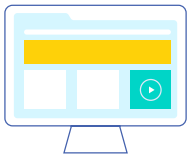
As APIs

Customers can access all actions of Courier through its APIs (through credentials and authentication) and integrate Courier into their pipelines and automation systems.



Powerful CLI

This is for customers who need or prefer to work on their terminal. It includes all the actions available through the API.



An intuitive UI experience

New upcoming features where a simple visual interface is provided to create Courier templates and access reports.

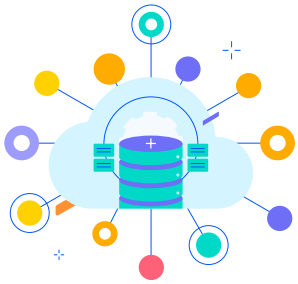
Job Definition Template

The Where

Definition: The section of the courier job definition through which the user tells where the job should be executed on the fleet.

```
"target":{
        "executionType": "sequential",
        "groups":[{
            "timeoutSeconds": 160,
            "batchSize": {
                "type": "number",
                "value": 3
            },
            "distributionMethod":"batching",
            "successCriteria": [{ "numRuns": { "type": "percent", "value": 100 }, "status": "success" }
            "nodeListType":"nodes",
            "nodeIdentifiers":["1b69900e-d095-4a7e-b57b-fe2a97ff2023","cb4c34f2-0caf-42ee-a99f-fdb99ac5
            },
```

This is defined using the courier job's target section. Multiple ways exist to target a courier job, a common one being a **node distribution group**.



Distribution Group

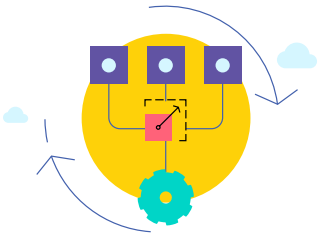
Each job can contain multiple node distribution groups or a composite of all node distribution groups.

- **List of node IDs:** The user provides a clear list of specific nodes for the job.

- **Node filter:** Reference for a node filter created in node management. Courier will resolve the nodes for the job for every instance from node management.

- **Node list:** Reference to a node list created in node management. Courier will resolve the nodes for the job for every instance from node management.

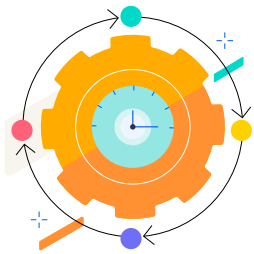
- **Node search query:** A text-based query string provided by the user for searching against node management. Courier will resolve the nodes for the job for every instance.



Execution Types

Sequential: Runs are performed on node distribution groups one after the other.

Parallel: Runs are performed on node distribution groups in parallel. Batching rules within each node distribution group will still be applicable.



Batching

Controls the number of nodes executing a job run within a node distribution group. Only the number of nodes specified in a batch size can execute runs together, and the rest in the node distribution group must wait for their batch.

Two methods can define the batch size:

- **Fixed number:** An integer value specifying the number of nodes accommodated into a batch. Example: 5 nodes per node distribution group.

- **Percentage:** A relative value based on the number of nodes within a node distribution group. Example: 10% of nodes per node distribution group.

Batching includes two types:

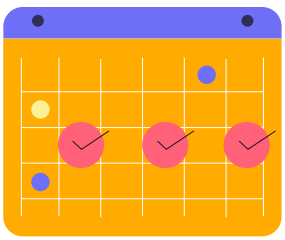
• **Fixed batch:** All nodes within a node distribution group are divided into batches as specified in the job definition. The batches will be executed sequentially.
Example: Batch size of 5 for node distribution group [a,b,c,d,e,f,g,h,i,j,k]. Job runs will be performed first on a,b,c,d,e together. After these 5 are complete, then f,g,h,i,j will execute together. Job run will happen on node k only after the completion of the second batch.

• **Rolling batches:** At any point, only the batch size of the nodes specified can be executed within a node distribution group. Nodes from within a node distribution group will be added to a batch when a node inside the batch completes the execution.
Example: A batch size of 5 for the node distribution group [a,b,c,d,e,f,g,h,i,j,k]. Job runs will be performed first on a,b,c,d and e together.

The When

Definition: The section of the courier job definition through which the user tells when the job should be executed on the fleet. Only one schedule can be used per job.
All schedules are maintained in UTC only.

```
"immediate": false,
"scheduleRule": "RRULE:FREQ=SECONDLY;INTERVAL=120;COUNT=2",
```



Types of schedules for Courier jobs are:

• **Immediate run:** The job must be run immediately by the nodes. Nodes will execute these runs immediately after their current run is completed.

• **Scheduled run:** The job must be run at the specified time and date. Nodes will execute the run on the specified time and date after their current run completes.
Example: Run this job on January 1, 2024 at 1 am.

• **Chronic runs:** The job must be run multiple times on the specified date(s) and time(s). Each occurrence of this schedule will be an instance of the job.
Examples: Every Monday from January 1 to May 1, 2024, and every day of April 2024 at 3 pm.

Schedule exceptions

These are no-go date(s) during which no job run should be picked up.
Ongoing runs can continue, however such instances will be marked as not performed.

Schedule exception types

There are two types of schedule exceptions:

Global: These are a list of exceptions to schedules maintained globally across all jobs. The job definition can opt-in to incorporate these into its schedule.
Example: Amazon can mark black Friday sale days as global exceptions so that no jobs run on the nodes and overload them.

Per job: These are exceptions to the scheduled maintenance per job. They can be added in addition to global schedule exceptions if that has been opted in.
Example: A courier job to apply OS patches every six months can have exceptions to not perform on banking holidays when there will not be anyone available to debug if something goes wrong.

Cancelled instances: All instances of a job the user has explicitly prevented from running will not be sent to the runners. Records related to completed instances will continue to be stored for retrieval.
Example: The legal department has updated the policies around maintaining version upgrades. Hence, the courier's job of performing automated upgrades must be stopped to accommodate the new system of upgrades.

Classification based on execution and status:

Past instances:	**Live instances:**	**Completed jobs:**
This is a term for referring to complete instances. This includes successful and failed cases.	Every job instance that has run (on any node) and is currently underway now of checking is called a live instance.	Any jobs with no instances left to run are labelled as completed jobs. This can include jobs that have been successful, failed and cancelled.

The What

Definition: The aspect of the job definition through which the actions and steps are specified, along with how they must be executed. A package can have multiple actions and steps. Each package is to be executed entirely for every run.

```
"actions": {
        "accessMode": "agent",
        "steps":
        [
            {
                "stepNumber": 1,
                "interpreter": {
                    "name": "chef/courier-interpreter/os/shell"
                },
                "command": [
                    "sleep 10"
                ],
                "inputs": {},
                "expectedInputs": { },
                "stepOutputFieldRules": {},
                "retryCount": 2,
                "failureBehavior": {
                    "action": "retryThenFail",
                    "retryBackoffStrategy": {
                        "name": "none",
                        "delaySeconds": 0,
                        "arguments": [1,3,5]
                    }
                },
                "limits": {},
                "conditions": []
            }
        ]
    },
"timeoutSeconds": 300
```

- **Action:** A compilation of everything that happens on a node before, during and after a job run. There can be multiple actions per run. Actions will always be performed sequentially as defined in the job.

  Example: I want to perform log4j scan using specific inspec skill version on nodes only when they have the required amount of memory

- **Step:** Every individual command execution to be performed on the node. There can be multiple steps with multiple commands for an action. Steps require interpreters to perform the execution irrespective of whether they are Chef commands or not

- **Skill name:** The name of the executable to be used for performing the step. If the skill or its corresponding interpreter is not available on the node at the time of execution the action will be considered as failed

- **Action:** A compilation of everything that happens on a node before, during and after a job run. There can be multiple actions per run. Actions will always be performed sequentially as defined in the job.
  Example: I want to perform log4j scan using specific inspec skill version on nodes only when they have the required amount of memory

- **Step:** Every individual command execution to be performed on the node. There can be multiple steps with multiple commands for an action. Steps require interpreters to perform the execution irrespective of whether they are Chef commands or not

- **Skill name:** The name of the executable to be used for performing the step. If the skill or its corresponding interpreter is not available on the node at the time of execution the action will be considered as failed

- **Version:** Any specific version(s) to be used when invoking the step's skill. Version is optional and if no value is provided then the runner will execute against whatever is the default version of the skill available on the node. Version can be specified in different forms:

  - **Exact version.** This is a single value, and the step will be performed on this version of the skill alone.
    Example: Inspec version 5 will execute the step on version 5 only.
  - **Range:** A minimum and maximum value within which any version available can be used for executing the step.
    Example: Inspec version 4 to 6 will execute the step on Inspec 4/5/6, whichever is available on the node.
  - **Wildcard:** This is denoted by the character *. Any version of the skill matching the wildcard will be used for executing the step.
    Example: Inspec 4.x will execute the step on any version of 4 available on the node.

- **Payload:** The content/command to be executed by the skill.
  Examples: List of commands to be run by OS command skill, the cookbook to be run by Infra client.

- **Precondition:** Criteria that the node must meet for every action before execution of the step. Action will retry the number of times mentioned in retries until it meets preconditions before marking it as failed action. There can be multiple preconditions per action.

- **Limit:** Outer confines that an action cannot exceed. The action will stop execution of the step when the limit is crossed. There can be multiple limits per action.
  Example: Do not run the step in more than one core.

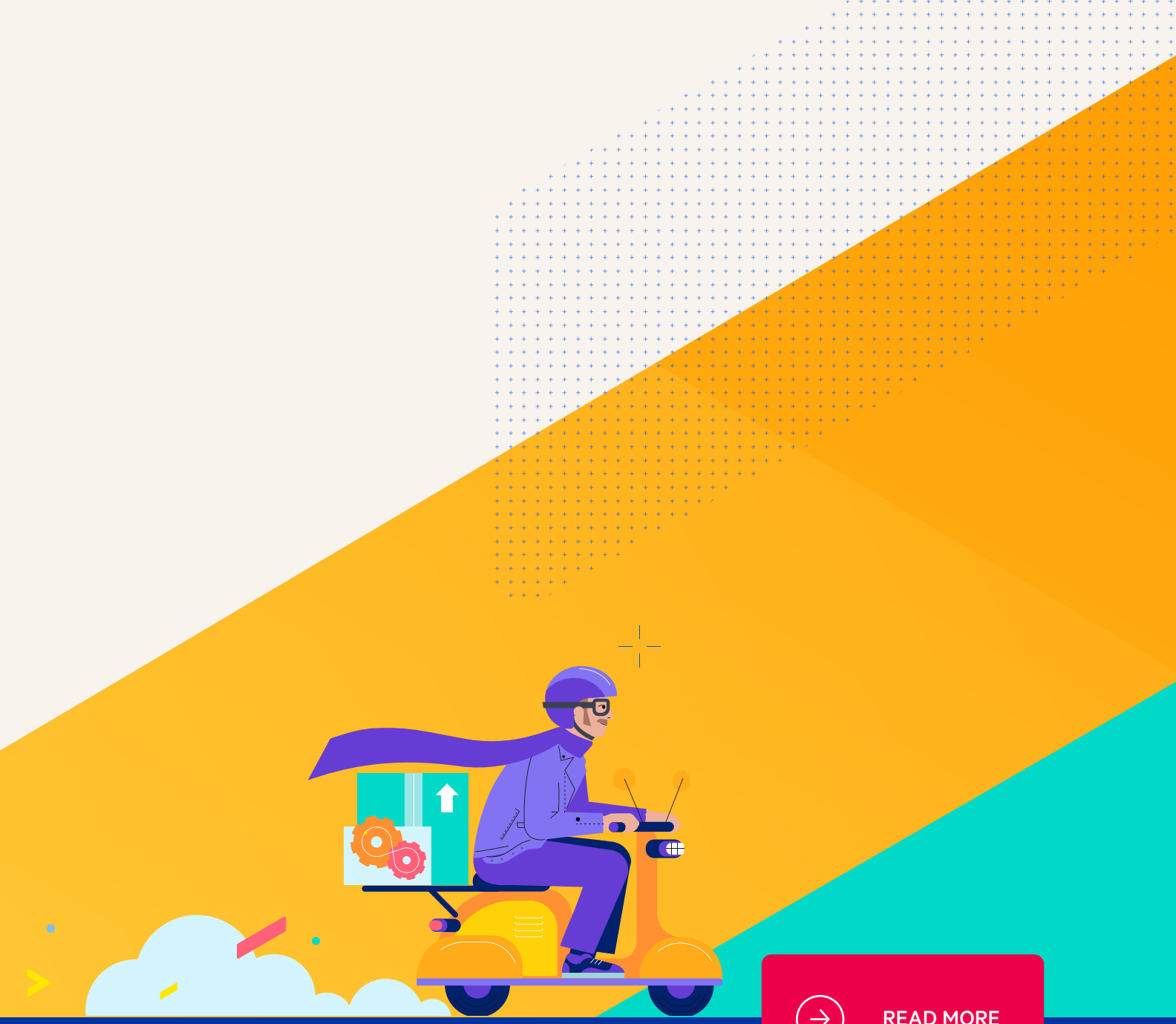
- **States:** An action can be of multiple states depending on how the step executed:
  - **Completed action:** An action is deemed completed after the step finishes execution.
  - **Successful action:** An action is labeled as successful if it created the intended outcomes.
  - **Failed action:** An action is labeled as failed if it faced an error upon execution/ timed out/did not meet a success criterion.
  - **Success criteria:** Every action can have a custom definition of how to check for its success. The default success criteria for actions dictate that there are no errors in its output when the step is executed.
    Examples:
    1. A file operation to create a certain folder structure is called a successful operation if the desired file folders are created.
    2. A compliance scan can be considered successful only if 95% of inspec profile controls succeed.
    3. An API call should contain text that matches a regex expression.
  - **Optional actions:** Actions that will be only tracked for completion and not for success/failure.
    Example: Reindexing the database is optional before exporting into a dump in the next action.

- **Inputs:** Any specific values that need to be provided by the step during run time for proceeding with the execution.
  Examples: Yes/No for file remove steps, cookbook location for client run, local mode options.

- **Actions chaining:** Every job can have multiple actions. These actions are executed sequentially by default. However, the sequencing can be conditional. Each action can have a condition on the states to check for against the previous action.
  Example: Perform a remediation action only if the previous compliance scan action failed. Execute an anti-virus scan action after completion of previous action.

- **Retries:** Every action can be re-attempted to completion if they have failed. By default, no action is retried if failed.
  Example: Action to stop a background process can have retries as 5 since it might not stop the first time.

- **Job run states:**
  - Completed: A run is deemed completed after all the actions complete.
  - Successful: A run is labeled as successful if all the actions succeed as per their criteria.
  - Failed runs: A run is labeled as failed if any of the actions failed to meet their success criteria. A run is also deemed as failed if the node is unreachable.



READ MORE

About Progress

Progress (Nasdaq: PRGS) empowers organizations to achieve transformational success in the face of disruptive change. Our software enables our customers to develop, deploy and manage responsible AI-powered applications and experiences with agility and ease. Customers get a trusted provider in Progress, with the products, expertise and vision they need to succeed. Over 4 million developers and technologists at hundreds of thousands of enterprises depend on Progress. Learn more at www.progress.com

Worldwide Headquarters

Progress Software Corporation
15 Wayside Rd, Suite 400, Burlington, MA 01803, USA
Tel: +1-800-477-6473

- facebook.com/getchefdotcom
- twitter.com/chef
- youtube.com/getchef
- linkedin.com/company/chef-software
- learn.chef.io
- github.com/chef
- twitch.tv/chefsoftware

Progress