Information points reflect key performance metrics and custom metrics for methods and code data points that you configure. You can define information points for PHP applications as described in this topic.

**About PHP Information Points**

To create an information point for PHP, you first define the information point using JSON notation.

In the definition, you specify the method for the information point and optionally a point in the code that you want to capture as a custom metric, such as a parameter value or return value.

After creating the definition, add an information point in the More > Information Points page in the Controller UI. When adding the information point, choose PHP as the agent type and then paste the definition into the information point text box, as shown in the following screenshot:
Essentially, the elements of the definition correspond to configuration options available in the Add Information Point dialog box for Java or .NET applications.

However, for PHP, the information point is defined in JSON syntax, and there are a few limits on configuration options. For one, class matching is limited to classname-based matches only. (Superclass matching or annotation matching, for example, are not available.) Also, you can only define a single match condition on a matched method. (A match condition refines the match of a method by testing a parameter or return value.)

If you do not configure a custom metric, the information point captures the generic KPIs for a matched method (response time, calls and calls per minute, errors or errors per minute). Custom metrics extend the information point by capturing the return or parameter value of a method, or the value of an invoked object.

**Before Starting**

To create information points in the Controller UI, you need to be logged in as a user with Configure Information Points permissions. See Roles and Permissions for more information.

**Method Match Conditions**

A match definition specifies the method associated with the value you want to capture as an information point. The match definition must specify the method, but it may also specify the containing class and a refining match condition. The match condition can test the value of a method parameter, return value, or the return value of an invoked object.
Use the following template to create a match definition:

```json
{
    "probe": {
        "phpDefinition": {
            "classMatch": {
                "type": "MATCHES_CLASS",
                "classNameCondition": {
                    "type": "EQUALS",
                    "matchStrings": [
                        "<class name>"
                    ]
                }
            },
            "methodMatch": {
                "methodNameCondition": {
                    "type": "EQUALS",
                    "matchStrings": [
                        "<method name>"
                    ]
                }
            }
        }
    }
}
```

Edit the JSON objects as follows:

- **classMatch** is optional. If you do not want to specify a class to match, delete the classMatch object from the phpDefinition object.
- If you are using the class match, set the value for the classNameCondition match string to the class name.
  - The class must be defined in a separate file from the file in which it is instantiated and invoked.
  - If your class names are defined in a namespace, escape the backslashes in the namespace with an escape backslash.
- **methodNameCondition** is required. Set the value for the methodNameCondition match string to the method name.

The following values are required for PHP information points:

- **classMatch type**: MATCHES_CLASS
- **classNameCondition type**: EQUALS
- **methodNameCondition type**: EQUALS

For example, the following JSON creates an information point on a class for which the class name equals `CheckoutManager` and the method name `processPayment`:
If creating an information point primarily to capture KPI metrics for the method, it is likely you will define the class for the method. However, if you are creating an information point to implement a code metric, you may only need to specify method name matching.

The following example shows a method match that tracks how many times a method is called:

```json
{
  "probe": {
    "phpDefinition": {
      "methodMatch": {
        "methodNameCondition": {
          "type": "EQUALS",
          "matchStrings": [
            "deleteCartItems"
          ]
        }
      }
    }
  }
}
```
Match Conditions

A match condition (defined by a `matchConditions` object) lets you refine the method matching criteria by specifying a match condition based on a parameter or return value or the value returned by an invoked object.

A match condition is optional, but there can be at most one for the information point. If you do not supply a match condition, the information point includes all invocations of the matched method.

The condition consists of a match type and the comparison operator that defines the data to be compared. The match types are:

- EQUALS
- NOT_EQUALS
- LT
- GT
- LE
- GE
- NOT
- AND
- OR

The comparison operator is an expression node that defines a left side (`lhs`) and a right side (`rhs`) of the operation. It contains:

- a type: ENTITY, STRING, or INTEGER
- a value, which is an `entityValue`, `stringValue`, or `integerValue`

If the type is ENTITY, the `entityValue has one of the following types: INVOKED_OBJECT, RETURN_VALUE, or PARAMETER. If the type of the `entityValue is PARAMETER, the parameterIndex indicates the 0-based numeric index of the parameter on which to match. So if the method takes two parameters, for example, the first being a string and second an integer, a parameterIndex of 0 matches against the string and a parameterIndex of 1 matches the integer value.

The information point in the following example only matches invocations of the `processPayment` method in which the second parameter (at parameter index 1) equals VISA.
Metric Definitions

You can define custom metrics for data captured by an information point using the `metricDefinitions` object in your information point definition. Custom metrics appear in the Metric Browser and in the dashboard for the information point. You can create health rules based on a custom metric and retrieve custom metric values using the AppDynamics REST API.

There can be multiple custom metrics for a single information point. For example, one metric might be based on the average and one on the accumulated (sum) of the information point's values.

A `metricDefinitions` object consists of one or more definitions, each having the following structure:

- a name
- a rollup type (AVERAGE or SUM)
- data, which consists of a type, (ENTITY, STRING, or INTEGER) and a value (entityValue, stringValue, or integerValue)

If the type is ENTITY, the entityValue has a type, which is INVOKED_OBJECT, RETURN_VALUE, or PARAMETER. If the type of the entityValue is PARAMETER, the zero-based parameterIndex indicates the parameter on which to base the match. A return
value or parameter metric cannot be an array.

For example, the following metricDefinitions object defines two custom metrics: VisaTotal, which reports the sum of the Visa payments processed and VisaAverage, which reports the average value of the Visa payments processed.

```json
"metricDefinitions": [
{
    "name": "VisaTotal",
    "rollup": "SUM",
    "data": {
        "type": "ENTITY",
        "entityValue": {
            "type": "RETURN_VALUE"
        }
    }
},
{
    "name": "VisaAverage",
    "rollup": "AVERAGE",
    "data": {
        "type": "ENTITY",
        "entityValue": {
            "type": "RETURN_VALUE"
        }
    }
}
]
```

Sample JSON Information Point Configuration

This is an example that you can copy and paste into the JSON text field in the PHP information point window. It produces two metrics, named VisaTotal and VisaAverage.

```json
{
    "probe": {
        "phpDefinition": {
            "classMatch": {
                "type": "MATCHES_CLASS",
                "classNameCondition": {
                    "type": "EQUALS",
                    "matchStrings": [
                        "CheckOutManager"
                    ]
                }
            }
        }
    }
}
```
"methodMatch": {
  "methodNameCondition": {
    "type": "EQUALS",
    "matchStrings": [
      "processPayment"
    ]
  },
  "matchCondition": {
    "type": "EQUALS",
    "comparisonOp": {
      "lhs": {
        "type": "ENTITY",
        "entityValue": {
          "parameterIndex": 1,
          "type": "PARAMETER"
        }
      },
      "rhs": {
        "type": "STRING",
        "stringValue": "VISA"
      }
    }
  }
},
"metricDefinitions": [
  {
    "name": "VisaTotal",
    "rollup": "SUM",
    "data": {
      "entityValue": {
        "type": "RETURN_VALUE"
      },
      "type": "ENTITY"
    }
  },
  {
    "name": "VisaAverage",
    "rollup": "AVERAGE",
    "data": {
      "type": "ENTITY",
      "entityValue": {
        "type": "RETURN_VALUE"
      }
    }
  }
]


```json
}
]
}
```