```python
import json
import boto3

# Function to parse evenet payload and extract relevant information
def event_parser(event):
    event_dict = dict()
    event_dict['resultToken'] = event['resultToken']
    event_dict['ruleParameters'] = json.loads(event['ruleParameters'])
    changed_event = json.loads(event['invokingEvent'])
    event_dict['orderingtimestamp'] = changed_event['configurationItem']['configurationItemCaptureTime']
    event_dict['lambda_arn'] = changed_event['configurationItem']['ARN']
    return event_dict

# Function to check compliance by retrieving lambda details
def evaluate_compliance(lambda_arn):
    lambdaclient = boto3.client('lambda')
    lambda_details = lambdaclient.get_function(
        FunctionName=lambda_arn
    )
    if "Environment" in lambda_details['Configuration']:
        if "KMSKeyArn" in lambda_details['Configuration']:
            return "COMPLIANT"
        else:
            return "NON_COMPLIANT"
    else:
        return "NOT_APPLICABLE"
```

```python
# Function to build the final message to post to AWS Config Rule
def build_config_message(compliance_status, lambda_arn, orderingtimestamp, resultToken):
    config_client = boto3.client("config")
    if compliance_status == "COMPLIANT":
        config_client.put_evaluations(
            Evaluations=[
                {
                    'ComplianceResourceType': 'AWS::Lambda::Function',
                    'ComplianceResourceId': lambda_arn,
                    'ComplianceType': compliance_status,
                    'Annotation': 'Lambda has a Customer Managed Key for its variable encryption',
                    'OrderingTimestamp': orderingtimestamp
                },
            ],
            ResultToken=resultToken,
            TestMode=True
        )

    elif compliance_status == "NON_COMPLIANT":
        config_client.put_evaluations(
            Evaluations=[
                {
                    'ComplianceResourceType': 'AWS::Lambda::Function',
                    'ComplianceResourceId': lambda_arn,
                    'ComplianceType': compliance_status,
                    'Annotation': 'Lambda does not have a Customer Managed Key for its variable encryption',
                    'OrderingTimestamp': orderingtimestamp
                },
            ],
            ResultToken=resultToken,
            TestMode=True
        )
```

There is no except handling. The parse function should have some sort of exception handling for each data/parameter that it needs. This is true for majority of the code written. **What does good look like? Comments, Exception handling. Code is readable**

```python
def evaluate_parameters(rule_parameters):
    # if parameter is given, check whether it is subnet Id or not, if it's not a subnetId then ignore it as it is an optional paramet
    if "subnetId" not in rule_parameters:
        return {}

    #split the parameter by delimiter "," and remove whitespace, then validate the subnetId
    subnet_id_list = [each_subnet.strip() for each_subnet in rule_parameters['subnetId'].split(',')]
    # to remove the empty item from the list (caused due trailing comma ',' in the parameter)
    subnet_id_list = list(filter(None, subnet_id_list))

    for each_subnet in subnet_id_list:
        if "subnet-" not in each_subnet:
            raise ValueError('Invalid value for the parameter "subnetId", Expected Comma-separated list of Subnet ID\'s that Lambda f
```

Making calls to external resources and combining Config rule logic is bad practice. There is no exception handling, the validation logic while works, doesn't take into account for any exceptions. Connection to boto3, has no exception handling. **What does good look like? Comments, Exception handling. Code is readable**

```python
def evaluate_compliance(event, configuration_item, valid_rule_parameters):
    # if "vpcConfig" is not present, then lambda function is outside the VPC
    if 'vpcConfig' not in configuration_item['configuration']:
        return build_evaluation_from_config_item(configuration_item, 'NON_COMPLIANT', annotation='This Lambda Function is not in VPC.')

    # if "vpcConfig" exists but no subnet is present, this scenario is possible if the lambda function was previously part of a VPC.
    if not configuration_item['configuration']['vpcConfig']['subnetIds']:
        return build_evaluation_from_config_item(configuration_item, 'NON_COMPLIANT', annotation='This Lambda Function is not in VPC.')

    # if no input parameter provided then return COMPLIANT
    if not valid_rule_parameters:
        return build_evaluation_from_config_item(configuration_item, 'COMPLIANT')

    # compare the subnets of lambda with the input parameter using set operation
    if set(configuration_item['configuration']['vpcConfig']['subnetIds']) - set(valid_rule_parameters):
        return build_evaluation_from_config_item(configuration_item, 'NON_COMPLIANT', annotation='This Lambda Function is not associated with the subnets

    return build_evaluation_from_config_item(configuration_item, 'COMPLIANT')
```

Example: connection to boto3, with exception handling

```python
try:
    AWS_CONFIG_CLIENT = get_client('config', event)
    if invoking_event['messageType'] in ['ConfigurationItemChangeNotification', 'ScheduledNotification', 'OversizedConfigurationItemChangeNotification']:
        configuration_item = get_configuration_item(invoking_event)
        if is_applicable(configuration_item, event):
            compliance_result = evaluate_compliance(event, configuration_item, valid_rule_parameters)
        else:
            compliance_result = "NOT_APPLICABLE"
    else:
        return build_internal_error_response('Unexpected message type', str(invoking_event))
except botocore.exceptions.ClientError as ex:
    if is_internal_error(ex):
        return build_internal_error_response("Unexpected error while completing API request", str(ex))
    return build_error_response("Customer error while making API request", str(ex), ex.response['Error']['Code'], ex.response['Error']['Message'])
except ValueError as ex:
    return build_internal_error_response(str(ex), str(ex))
```

```
else:
    config_client.put_evaluations(
        Evaluations=[
            {
                'ComplianceResourceType': 'AWS::Lambda::Function',
                'ComplianceResourceId': lambda_arn,
                'ComplianceType': compliance_status,
                'Annotation': 'Lambda does not have any environment variables',
                'OrderingTimestamp': orderingtimestamp
            },
        ],
        ResultToken=resultToken,
        TestMode=True
    )
```

```
# Main Lambda event handler and logging
def lambda_handler(event, context):
    try:
        event_dict = event_parser(event)
    except Exception as error_message:
        print("Unable to parse invoking dictionary")
        print(error_message)

    try:
        compliance_status = evaluate_compliance(event_dict['lambda_arn'])
    except Exception as error_message:
        print("Error while evaluating compliance")
        print(error_message)

    try:
        build_config_message(compliance_status, event_dict['lambda_arn'], event_dict['orderingtimestam
                             event_dict['resultToken'])
    except Exception as error_message:
        print("Error posting config rule message")
        print(error_message)

    print('KEY DATA')
    print(event_dict)
```

Error messages will not show which lambda resource is failing , the error message will simply say “Unable to parse invoking dictionary. Consider what would happen when the same rule check fails for 100s of lambdas across many accounts. Someone troubleshooting will not know what occurred? **If enabling these logs will slow the logic down, have necessary logging that is commented out, and when required can be enabled and disabled as required, so the customer doesn’t have to create that logic.**

The Unit case tests were happy path, example the following, the excepted value is NONE, so no matter what you pass, the case will pass. The unit test should test for actual failures when certain conditions are not met, like Lambda ARN doesn’t match, and what was the reason in detail as to why it failed.

```
# Test to check for build_config_function
def test_build_config_message(self):
    expected_value = None
    with patch('lambda_env_check.boto3.client') as mock_boto3_client:
        mock_boto3_client.return_value.put_evalutions.return_value = None
        returned_function = lambda_env_check.build_config_message(self.compliance_status, self.lambda_arn,
                                                                  self.orderingtimestamp, self.resultToken)
    self.assertEqual(expected_value, returned_function, msg="Build config cannot be posted work")
```

A good unit test

```
# common scenarios
def test_invalid_parameter_value(self):
    invoking_event = generate_invoking_event(self.lambda_inside_vpc)
    response = RULE.lambda_handler(build_lambda_configurationchange_event(invoking_event, rule_parameters=self.rule_invalid_parameter), {})
    assert_customer_error_response(self, response, 'InvalidParameterValueException', 'Invalid value for the parameter "subnetId", Expected Comma-separated list of S

# Scenario 2
def test_empty_parameter_value(self):
    invoking_event = generate_invoking_event(self.lambda_inside_vpc)
    response = RULE.lambda_handler(build_lambda_configurationchange_event(invoking_event, rule_parameters=self.rule_empty_parameter_value), {})
    resp_expected = []
    resp_expected.append(build_expected_response('COMPLIANT', 'test_function', 'AWS::Lambda::Function'))
    assert_successful_evaluation(self, response, resp_expected)
```